

f.i.d.e.[™]

Fuzzy
Inference
Development
Environment

Contents

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manuals distributed with an Apronix product or in the media on which a software product is distributed, Apronix will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apronix or an authorized Apronix dealer during the 90-day period after you purchased the software. In addition, Apronix will replace damaged software media and manuals for as long as the software product is included in Apronix's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apronix dealer for program coverage and details. In some countries the replacement period may be different; check with your authorized Apronix dealer.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUALS, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apronix has tested the software and reviewed the documentation, APRONIX MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.

IN NO EVENT WILL APRONIX BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apronix shall have no liability for any programs or data stored in or used with Apronix products, including the costs of recovering such programs or data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apronix dealer, agent or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Microsoft Windows, Microsoft C, and Microsoft DOS are registered trademarks of Microsoft Corporation, Inc. Turbo C and Borland C are registered trademarks of Borland International, Inc. IASM is a registered trademark of P&E Microcomputer Systems, Inc.

Fide and Apronix are trademarks of Apronix, Inc.

Part I

Fide Tutorial

Chapter 1	Fuzzy Inference	1
Introduction.....		1
Fuzzy Sets		2
Labels		5
Fuzzy inference rules.....		7
Inference Process.....		11
Fuzzification.....		12
Rule Evaluation		13
Defuzzification.....		15
Summary		16
Inference Unit with Several Inputs.....		18
TVFI Versus Mamdani's Inference Methods		21
Output membership function		21
Singletons associated to output labels		28
Variations of fuzzy inference unit		31
More Logical Operators		31
More on Defuzzification.....		31
Fuzzy Systems		33
Chapter 2	Developing Fuzzy Inference Units	37
Writing and Compiling Source Code		37
Debugging With The Fide Tracer		45
Debugging With The Fide Analyzer		52
Using The Fide Simulator.....		58
Real-time Code Generator.....		61
Chapter 3	Developing A Fuzzy Inference System	65
Fide Composer and Fide Library.....		65
Using the graphics editor		68
Making an executable file		79
Using The Data Flow Viewer.....		82

Simulating System Behavior.....	84
Part II User's Guide	
Introduction	93
What Is Fide?.....	93
Hardware and Software Requirements.....	94
Fide Package.....	94
Using The Manuals.....	95
Fide Quick Start Guide.....	95
Fide User's Manual.....	95
Fide Tutorial.....	95
Fide User's Guide.....	96
Fide Reference Manual.....	96
FIL Reference.....	96
FCL Reference.....	96
Apronix Run Time Library Reference.....	97
Typefaces Used In Fide Manuals.....	97
How to Contact Apronix.....	98
BBS.....	98
FAX.....	98
Voice.....	99
Installing Fide.....	99
Setup.....	99
Hardware Key.....	102
Fide Basics.....	103
Starting Fide.....	103
From DOS.....	103
From Windows.....	103
Exiting Fide.....	105
Fide User's Guide Reference Conventions.....	105
For windows.....	105
For menu commands.....	106
Chapter 1 Creating Source Files	109

File Menu Commands.....	109
New.....	110
Open.....	112
Close.....	112
Save.....	112
Save As.....	113
About Fide.....	113
Exit.....	114
Edit Menu Commands.....	114
Undo.....	115
Cut.....	115
Copy.....	115
Paste.....	115
Clear All.....	115
Go To.....	116
Search.....	116
Find.....	117
Replace.....	118
Next.....	118
Keyboard and Mouse Editing Functions.....	119
Window Menu Commands.....	119
Title.....	120
Cascade.....	121
Arrange Icons.....	122
Clear All.....	122
Windows List.....	123
Help.....	125
Chapter 2 Editing Membership Functions	125
MF-Edit Command.....	126
Select Variable.....	127
Input Label.....	128
Grid Button.....	128
Modify Button.....	129
Label Dialog Box.....	129

Cancel Button.....	130
Operation Button.....	130
Draw Button.....	130
Set Margin Button.....	131
Polygon Button.....	134
Graphics Editing.....	134
Text Entry Editing.....	135
Function Push Buttons.....	136
Output Label.....	138
TVFI Inference Method.....	138
Mamdani Inference Method.....	139
Chapter 3 Compiling Source.....	141
Compile Menu Commands.....	141
Source File.....	141
Data File.....	142
Chapter 4 Debugging And Analyzing.....	147
Debug Command.....	147
Fide Debugger.....	148
Target.....	148
Tools.....	149
Tracer.....	150
Select Variable.....	150
Set Input Value.....	150
Input Labels.....	151
Label Buttons.....	152
Grid.....	153
Modify.....	154
Draw Button.....	154
Operation Button.....	155
Set Margin Button.....	156
Polygon Button.....	158
Graphics Editing.....	158
Text Entry Editing.....	159
	160

Function Push Buttons.....	162
Execute.....	162
Output Label.....	163
Trace.....	164
Output Membership Function Window.....	164
Select Output Label.....	165
Select Rules.....	167
Jump To Source.....	168
Analyzer.....	169
Select Variables.....	169
Set Input Value.....	170
Execute.....	171
Display.....	173
Single Input Variable.....	173
Two Input Variables.....	174
Analyzer Views.....	175
All Views.....	175
Surface.....	175
Rotate Button.....	177
Trace Button.....	178
Exit Button.....	178
Contour.....	178
Trace.....	179
Rotate Counter Clockwise.....	179
Rotate Clockwise.....	180
Exit Button.....	180
Cross Section.....	180
Redraw Button.....	181
Exit Button.....	181
Simulator.....	182
Select Variables.....	182
Set Input Value.....	184
Execute.....	185
Display.....	185
Trace Button.....	186
Grid Button.....	186
Exit Button.....	186

<< Button.....	187
>> Button.....	187
< Button.....	187
> Button.....	187
Chapter 5 Generating Run Time Code	189
RTC Menu Commands	
Motorola Chip Support	189
MC6805.....	191
MC68HC05.....	191
MC68HC11.....	191
	192
Chapter 6 Composing A System	193
Composer Functions	
File.....	194
New Graphics.....	195
New Text.....	195
Open As Graphics.....	196
Fide Inference Unit (FIU).....	197
Fide Operation Unit (FOU).....	198
Fide Execution Unit (FEU).....	199
Fide Input Node.....	199
Fide Output Node.....	199
Open As Text.....	200
Close.....	200
Save As Graphics.....	202
Save As Text.....	202
Exit.....	203
Edit.....	203
Undo.....	203
Cut.....	204
Copy.....	204
Paste.....	205
Clear All.....	205
Search.....	205

Find.....	205
Replace.....	206
Next.....	207
Graphics.....	207
Draw Column.....	208
Draw Unit.....	209
Draw Variable.....	210
Draw Line.....	211
Erase.....	213
Spy.....	213
Zoom Out.....	214
Zoom In.....	214
Make.....	215
C Source Code.....	215
Make For Execution.....	216
Make For Data Flow.....	218
Make For Simulation.....	218
Run.....	219
Set Value.....	219
Select Input File.....	221
Set Steps.....	222
Run.....	222
Debug.....	223
Display Data Flow.....	224
Display Simulation.....	225
Set Range.....	225
Display.....	226
<< Button.....	227
>> Button.....	227
Trace.....	227
Window.....	228
Tile.....	228
Cascade.....	229
Close All.....	230
Windows List.....	230
Help.....	231
Tool Buttons.....	231

Appendix A Working Files	233
Compiler	233
Real Time Code Generator	234
Composer	234
Appendix B Messages	235
Compiler	235
Fatal Errors	235
Errors and Warnings	235
MF-Editor	242
Debugger	242
Run Time Code	246
Composer	250
.....	257

Index

Introduction

Fuzzy inference is a new methodology of problem solving, with applications in information processing and controls. This chapter introduces you to basic concepts of fuzzy inference and fuzzy inference systems.

Let us start with an example of temperature control. You learned how to adjust the temperature in your room by being told rules like the following:

*If the temperature is higher than what you desire, turn on the cool air fan;
if the temperature is lower than desired, turn on the warm air fan;
The further the temperature from desired, the faster the fan should run.
However, if the temperature is at desired, turn off the fan(s).*

The above rules are different from the ones taught in engineering textbooks because terms like "further" and "faster" are fuzzy (not precisely defined). Even terms like "is at desired" should not be regarded as an equality. We will call such terms fuzzy concepts, and such rules fuzzy rules.

Can we utilize such rules in engineering? The answer is "yes". How? To answer this question is the main topic of this document. And the key to the answer is Fuzzy Logic.

Fuzzy logic provides a systematic interpretation of fuzzy rules.

In fuzzy logic, a system based on fuzzy rules is regarded as a mechanism generating signals at its output ends in response to signals received at its input ends. Such a system is called a fuzzy inference unit, or inference unit for short. More precisely, an inference unit consists of a set of input variables, a set of output variables and a mechanism implementing a set of inference rules.

Suppose we are trying to establish a temperature controller based on the above fuzzy rules. We should have a temperature sensor and a mechanism for setting desired temperature and calculating the (algebraic) difference between the real and the desired temperatures. The signal obtained from the equipment is called

the temperature difference and is noted by temperature_diff for short. This is the input variable of the controller.

Two output variables are engaged: the speed of a hot air fan and the speed of a cool air fan, noted hot_fan_speed and cool_fan_speed respectively.

Therefore, the controller is a fuzzy inference unit with the input and output variables as shown in Figure 1.1.

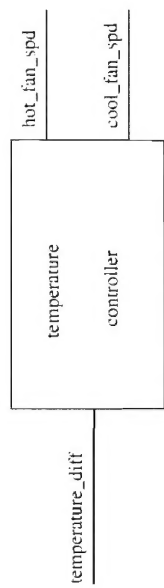


Figure 1.1 A temperature controller as an inference unit with an input variable and two output variables

Fuzzy Sets

In terms of temperature_diff, hot_fan_speed and cool_fan_speed, the fuzzy inference rules should be reworded as follows:

```
if temperature_diff is PositiveBig
then cool_fan_speed is high;
if temperature_diff is zero
then hot_fan_speed is zero;
if temperature_diff is zero
then cool_fan_speed is zero;
if temperature_diff is zero
then cool_fan_speed is zero;
if temperature_diff is NegativeMedium
then cool_fan_speed is medium;
```

and so on. The terms "PositiveBig", "Zero", "NegativeMedium" etc. are fuzzy because they are not well-defined in the sense of traditional physics and mathematics.

In traditional mathematics, a well-defined adjective (used as a predicate) such as "positive" and "negative", when applied to a variable, is regarded as a set in the space (or universe of discourse) of the values for the variable. For example, the space of temperature_diff is, say, (-110, 100) in Fahrenheit, and the adjective "positive" refers to the interval (0, 100), which is a set in the space. Such a set is also characterized by a characteristic function or membership function. By definition, the characteristic function of a set A in space X, shown in Figure 1.2, is the function such that:

$$f(x) = 1 \quad \text{if } x \text{ is a member of } S$$
$$f(x) = 0 \quad \text{if } x \text{ is not a member of } S$$

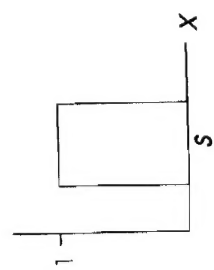


Figure 1.2 A membership function of a set S in space X is a function whose value is 1 at the points of S and 0 elsewhere.

Similarly, in fuzzy logic, the adjectives are regarded as fuzzy sets in the space of the values for the variable. By definition, a fuzzy set A in space X is characterized by a membership function f such that for any point x in X , $f(x)$ is a number between (and including) 0 and 1. The number is said to be the membership degree or grade of x , x being a member of A. This number can be thought of as the truth value of the statement " x is (in) A". We will use the term "grade" in most instances because it is shorter.

For example, the fuzzy set of "PositiveBig" in the space of temperature_diff is characterized by the function illustrated in Figure 1.3. As is evident from the Figure, the grade of the statement "80 is a PositiveBig temperature difference" is 1 (100%). The grade of the statement "20 is a PositiveBig temperature

difference" is 0. And the grade of the statement "40 is a PositiveBig temperature difference" is 0.5 (50%).

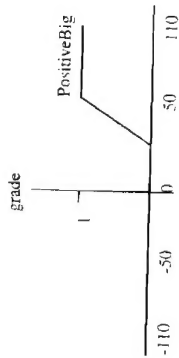


Figure 1.3 The membership function of fuzzy set 'PositiveBig'

We mentioned that a set in the traditional sense is also a fuzzy set. In particular, a singleton, or a set consisting of a single member is a fuzzy set. The unique member of a singleton is called the support point or the support value of the singleton. The membership function of a singleton vanishes everywhere in the space of the variable except at the support point, where the grade is 1 (Figure 1.4).

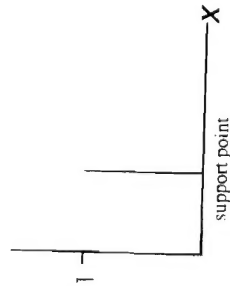


Figure 1.4 The membership function of a singleton vanishes everywhere except at the support point

Labels

Generally speaking, to each input variable of an inference unit, one or more fuzzy sets are associated. They are called the labels of the input variable, or the input labels if the input variable is not referenced.

In the temperature controller example, we will use the following labels:

- (L1) PositiveBigDiff,
- (L2) PositiveMediumDiff,
- (L3) PositiveSmallDiff,
- (L4) ZeroDifference,
- (L5) NegativeBigDiff,
- (L6) NegativeMediumDiff,
- (L7) NegativeSmallDiff.

Note: The notations (L1) to (L7) are not part of the labels, they are here only for reference purpose.

The corresponding membership functions are shown on Figure 1.5

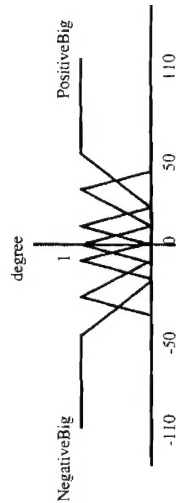


Figure 1.5 Labels of Temperature_diff are associated with fuzzy sets

On the other hand, the adjectives "high", "medium", etc., when applied to output variables, should not be regarded as membership functions. Rather they are regarded as actual (physical) values. For example, "high" may refer to 800 (rpm), medium may refer to 500 (rpm), and so on. Listed in the following table

are 5 labels applied to the output variables hot_fan_speed and cool_fan_speed (two variables may use the same labels).

Generally speaking, one or more labels are associated with each output variable, and each label has a corresponding physical value, called its support value (for reasons that will become clear below).

Table 1.1 Labels of hot_fan_spd are associated with support values

Label	support value
O1: zero	0
O2: low	200
O3: medium	500
O4: high	800
O5: veryhigh	1000

Note: The notations (O1) to (O5) are not part of the labels, they are here only for reference purpose.

The labels are illustrated in Figure 1.6.

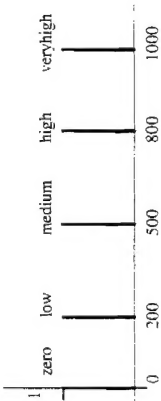


Figure 1.6 Labels of hot_fan_spd are associated with supporting values

The relationship of the input/output variables and their labels is illustrated in Figure 1.7.

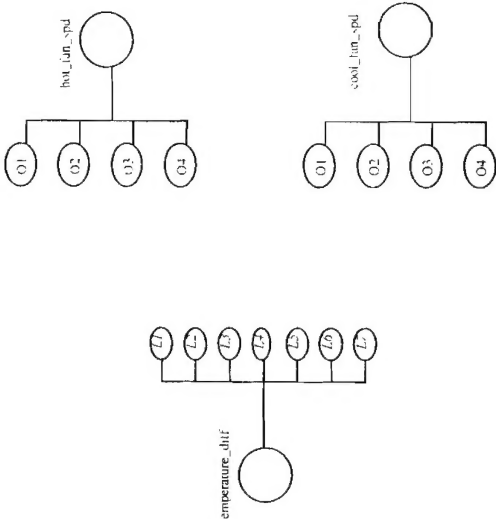


Figure 1.7 The variables and the labels of the temperature controller

Fuzzy inference rules

A fuzzy inference rule is a statement expressing the dependency between variables. It has the following format:

if <condition part> then <consequence part>

The <condition part> and <consequence part> are two assertions. For example, in the following rules:

if temperature_diff is PositiveBig
then hot_fan_speed is high

the <condition part> and <consequence part> are the assertions "temperature_diff is PositiveBig" and "hot_fan_spd is high" respectively. An assertion combining a variable and a label corresponds to a line between the variable and the label in Figure 1.7. In a similar way, a rule combining two assertions can be regarded as a channel from the first assertion to the second, through the two labels of the assertions, as illustrated in Figure 1.8.

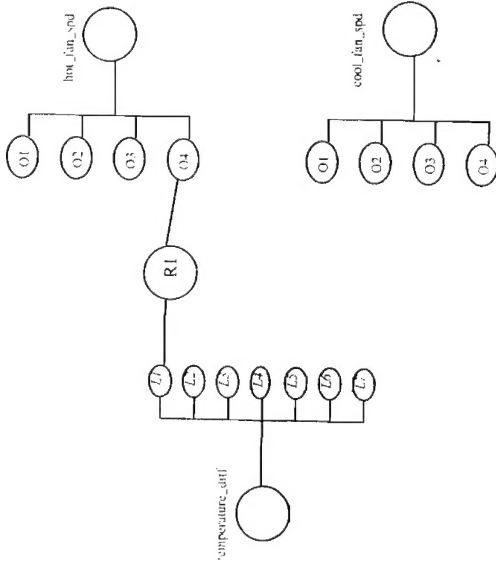


Figure 1.8 A fuzzy rule is a channel from an input label to an output label

The full set of 14 fuzzy inference rules of our temperature controller are listed below and illustrated with a chart as shown in Figure 1.9. Such a chart is called a (fuzzy) inference chart.

- (R1) if temperature_diff is PositiveBigDiff then hot_fan_spd is high;
- (R2) if temperature_diff is PositiveMediumDiff then hot_fan_spd is medium;
- (R3) if temperature_diff is PositiveSmallDiff then hot_fan_spd is low;
- (R4) if temperature_diff is ZeroDifference then hot_fan_spd is zero;
- (R5) if temperature_diff is NegativeSmallDiff then hot_fan_spd is zero;
- (R6) if temperature_diff is NegativeMediumDiff then hot_fan_spd is zero;
- (R7) if temperature_diff is NegativeBigDiff then hot_fan_spd is zero;
- (R8) if temperature_diff is PositiveBigDiff then cool_fan_spd is zero;
- (R9) if temperature_diff is PositiveMediumDiff then cool_fan_spd is zero;
- (R10) if temperature_diff is PositiveSmallDiff then cool_fan_spd is zero;
- (R11) if temperature_diff is ZeroDifference then cool_fan_spd is zero;
- (R12) if temperature_diff is NegativeSmallDiff then cool_fan_spd is low;
- (R13) if temperature_diff is NegativeMediumDiff then cool_fan_spd is medium;
- (R14) if temperature_diff is NegativeBigDiff then cool_fan_spd is high;

Note: 1. The notations (R1) to (R14) are not part of the rules, they are here only for reference purpose.
2. The output label "very high" is not used in these rules and is omitted in the chart.

Inference Process

The inference process is the internal mechanism producing output values through fuzzy rules for given input values.

An inference process involves three steps: fuzzification, rule evaluation, and defuzzification. Roughly speaking, fuzzification is the process of translating real world values of the input variable (input values, for short) to the grades of its input labels (input grades, for short). Rule evaluation is the process of obtaining the grades of output labels (output grades, for short) from input grades. The final step, defuzzification is translating output grades to output values, which are real world values of output variables (Figure 1.10).

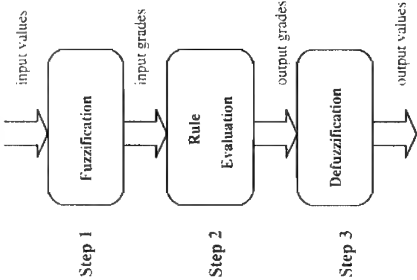


Figure 1.10 Inference process consisting of three steps

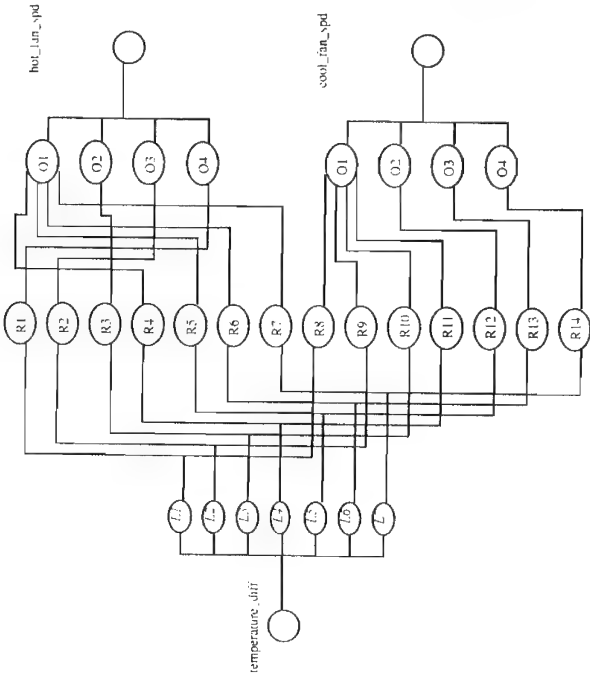


Figure 1.9 fuzzy inference chart of a temperature controller.

Fuzzification

Fuzzification converts input values into grades by means of (the membership functions of) input labels. A grade obtained by means of a given input label is called the grade of that input label .

For example, suppose the input value is temperature_diff=25. By means of membership functions (see Figure 1.5) we obtain the grades of the input labels, as listed below:

Table 1.2 Input grades of input labels

Input temperature_diff = 25	
Label	Grade
PositiveBigDiff	0
PositiveMediumDiff	0.75
PositiveSmallDiff	0.25
ZeroDifference	0
NegativeBigDiff	0
NegativeMediumDiff	0
NegativeSmallDiff	0

The fuzzification step is also illustrated in Figure 1.11, in which an input value is displayed at the left of the input variable, and the grades obtained in the fuzzification step are displayed to the right of the input labels.

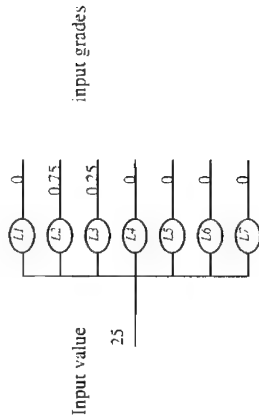


Figure 1.11 Fuzzification step generates grades of input labels

Rule Evaluation

The rule evaluation step computes the grade of each output label. This step, in general, involves rather complicated operations. Our example (included for tutorial purposes), however, is straightforward and does not demonstrate them. In our example, the rule are just straight channels through which the grades flow from the input label to the output labels. We illustrate the flow by writing the grades of the output labels obtained in this step, leaving the general case for deferred discussion (Figure 1.12)

Note: Occasionally, lines from several lines may meet at the same output label, as the two (O1)'s in Figure 1.12. This is caused by the fact that several rules share the same conclusion, as the rule (R8) through (R11). In this case, the highest grade coming to the output is written on the right of the label. The reason why the highest grade is taken will be discussed later. At present, we just regard it as an implementation of this principle: if among different alternatives, there is one that stands out as the most reasonable based on all available factors, then choose it.

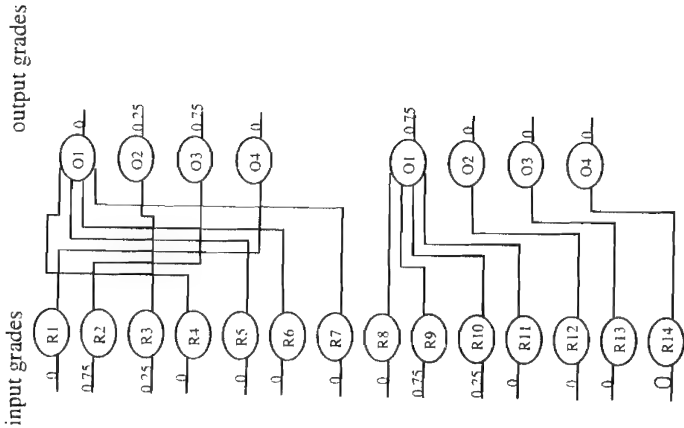


Figure 1.12 Rule Evaluation step generating output grades from input grades

The following is a list of the output grades obtained in this step:

Table 1.3 Output grades

Output variable: hot_fan_spd:	
Label	Grade
zero	0
low	0.25
medium	0.75
high	0

Output variable: cool_fan_spd:	
Label	Grade
zero	0.75
low	0
medium	0
high	0

Defuzzification

As expected, we have some fuzzy conclusion from the above step. This is clearly presented by the above table: we have a grade of 0.25 for the assertion that hot_fan_spd should be low and a grade of 0.75 for the assertion that the hot_fan_spd should be medium. A commonly used interpretation is to take these grades as a measure of confidence in the respective conclusions. The defuzzification step obtains a final decision on what the controller should do with the fans.

If we consider the grades as weights attached to the assertions, then a reasonable solution is to obtain the weighted average of the support values associated with the output labels as a way of arriving at a final decision. By means of output labels, an assertion such as "hot_fan_spd is low" is equivalent to "hot_fan_spd = 200", etc.. The weighted average of the hot_fan_spd is:

$$\begin{aligned} \text{hot_fan_spd} &= \\ & (0*0+0.25*200+0.75*500+0*800)/(0+0.25+0.75+0) \\ & = 425 \end{aligned}$$

Similarly, for the output variable cool_fan_spd, we have:

$$\begin{aligned} cool_fan_spd = & (0.775*0+0*200+0*500+0*800)/(0.75+0+0+0) \\ = & 0 \end{aligned}$$

These are the values generated by the internal mechanism of the fuzzy inference unit.

Summary

The fuzzy inference process consists of three steps:

- Fuzzification,*
- Rule Evaluation,*
- Defuzzification.*

Combining them together, we can show the sample data flow on the fuzzy inference chart as illustrated in Figure 1.13.

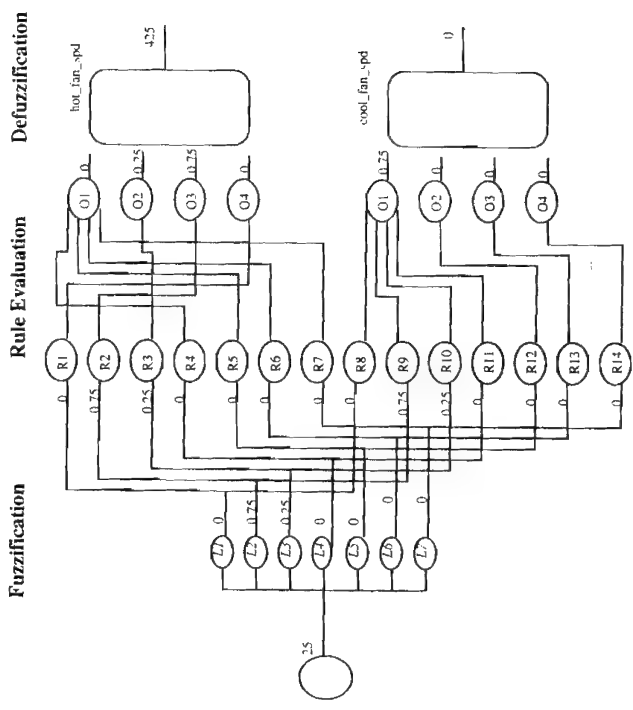


Figure 1.13 Summary of fuzzy inference chart with data flow

Inference Unit with Several Inputs

It is common that a fuzzy inference unit has more than one input variable and the fuzzy rules involve conditions on several input variables simultaneously.

Suppose, in the temperature controller example, we include another variable: `temperature_change -- the change of temperature.`

Associated with this variable are a set of 7 labels. Let them be the same as those for the variable `temperature_diff` for simplicity.

Now the control rules may take temperature trending into consideration. Let the rules be:

```

r-1: if temperature_diff is PositiveBigDiff and
    temperature_change is PositiveBigDiff
    then temperature_diff is medium
r-2: if temperature_diff is PositiveBigDiff and
    temperature_change is PositiveBigDiff
    then temperature_diff is low
r-3: if temperature_diff is PositiveBigDiff and
    temperature_change is PositiveMediumDiff
    then temperature_diff is zero
r-4: if temperature_diff is NegativeBigDiff and
    temperature_change is PositiveBigDiff
    then temperature_diff is zero
    
```

A set of rules like this is quite different from the simpler case discussed in the previous sections because the condition part of a rule has several assertions combined by the word "and". Therefore, the inference chart becomes more complicate as illustrated in Figure 1.14.

What will happen in the inference process for such a chart? Nothing new in the fuzzification step and the defuzzification step. But the rule evaluation step involves more operations.

Consider the rule (r-1). As we can see on Figure 1.14, two grade value will meet at the node representing the rule. Suppose they are:

```

assertion      grade
temperature_diff is PositiveBigDiff  0.75
temperature_change is PositiveBigDiff 0.12
    
```

What will be the grade obtained by combining the two assertions? The operation of obtaining the grade of "A and B" from the grade of "A" and the grade of "B" is called the logic AND operation or the conjunction. In our case, it is obvious that, because the temperature_change is not likely PositiveBigDiff, the rule (r-1) does not play an important role in this specific instance. A commonly accepted numeric implementation of the logic AND operation is to take the smaller grade of "A" and "B" as the grade of "A and B". This operation is called the minimum operation or minimization and is denoted by `min`:

$$\min(0.75, 0.12) = 0.12$$

On the other hand, we also see from the chart that the grades obtained from (r-3) and (r-4) meet at the node representing the output label (O-3). For instance, suppose the grades meeting here are:

```

grade      obtained from
0.88      (r-3)
0.32      (r-4)
    
```

What will be the grade obtained by combining the two rules? From the general logic consideration, a combination of two rules "if A then P" and "if B then P" is equivalent to a single rule "if A or B then P" and the operation of obtaining the grade of "A or B" from the grade of "A" and the grade of "B" is called the logic OR operation or the disjunction. In our case, the high confidence we have in the rule (r-3) indicates that the grade of the corresponding output label should be high also. A commonly accepted numeric implementation of the logic OR operation is to take the larger grade of "A" and "B" as the grade of "A or B". This operation is called the maximum operation or maximization and is denoted by max:

$$\max(0.88, 0.32) = 0.88$$

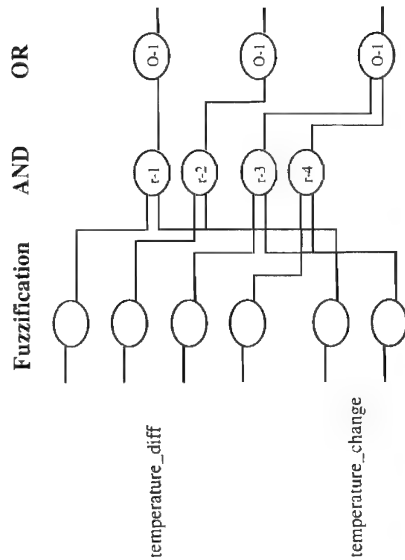


Figure 1.14 An inference chart with 2 input variables, showing some flows meet at nodes representing rules as well as nodes representing output labels. The fuzzy operations at these nodes are logical AND (conjunction) and logical OR (disjunction) respectively.

TVFI Versus Mamdani's Inference Methods

At the first glance, the input label and the output labels are not treated equally; the input labels are interpreted in terms of membership functions or fuzzy sets, but the output labels are interpreted in terms of the support values. Can we interpret the output labels in terms of membership functions as well? The answer is "yes", but this raises a new topic in the inference process, the inference method.

Actually, the inference process we have discussed so far is one of the possible mathematic models of fuzzy inference, called TVFI method. Another well-known model is the Mamdani's method, in which the output labels are regarded as membership functions rather than support values. Mamdani's method and its application will be discussed here.

Output membership function

With Mamdani's method, the expression "output to a variable (at the output end of the inference unit)" is regarded as a fuzzy set, and the output value is obtained by performing defuzzification on the fuzzy set. The membership function of the fuzzy set is said to be the output membership function of the variable. The output membership functions are generated by performing a special operation called aggregation (Figure 1.15).

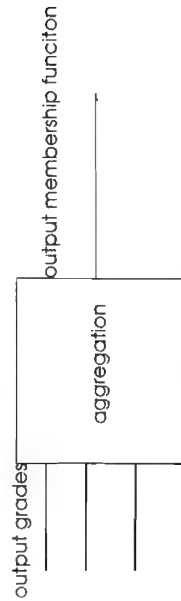


Figure 1.15 Aggregation is an operation generating output membership functions based on the output grades

Let us review the inference process. Suppose that, instead of support values, we have redefined the output labels as membership functions. The fuzzification step will not be influenced by the output labels. The rule evaluation step, which assign grades to the output labels, will remain the same. Aggregation is then inserted here, between the rule evaluation step and the defuzzification step, generating the output membership function of each output variable based on the grades of its labels and the membership functions associated with the labels.

The following numerical example shows how aggregation works.

Suppose, in the temperature controller example, the output labels of the variable `hot_fan_spd` are defined as membership functions illustrated in Figure 1.16.

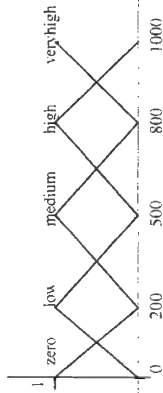


Figure 1.16 Membership functions associated to the output labels

The membership functions are denoted $zero(x)$, $low(x)$, $medium(x)$, $high(x)$ and $veryhigh(x)$, respectively.

Let the desired output value of the variable be v , and $g(x)$ be the output membership function. By definition, for a given a in the space of the variable, $g(a)$ is the degree of membership of a belonging to the fuzzy set "output to the variable" or " a is the output".

Suppose the following grades are obtained after the rule evaluation:

label	grade
zero	0.25
low	0.25
medium	0.75
high	0.75

Mamdani's method regards the grades as the grades of the output value v belonging to the corresponding labels. For example, the grade of " v is low" is 0.25.

On the other hand, the grades of a given value (in the space of the variable) belonging to a label is given as the membership function associated with the label. For example, the grade of " a is low" is $low(a)$.

The major effect of the Mamdani's method is to interpret the statement that " a is the output" by the statement " a is (roughly) the same as v ", which is then interpreted as the following composed statement:

a is zero and v is zero, or
 a is low and v is low, or
 a is medium and v is medium, or
 a is high and v is high

Therefore, we have:

$g(a)$
 = grade of " a is the output"
 = grade of " a is (roughly) the same as v "
 = grade of " a is zero and v is zero, or
 a is low and v is low, or
 a is medium and v is medium, or
 a is high and v is high"
 = the maximum of the following:
 grade of " a is zero and v is zero"
 grade of " a is low and v is low"

grade of "a is medium and v is medium"
 grade of "a is high and v is high"
 = the maximum of the following:
 $\min(\text{grade of "a is zero", grade of "v is zero"})$
 $\min(\text{grade of "a is low", grade of "v is low"})$
 $\min(\text{grade of "a is medium", grade of "v is medium"})$
 $\min(\text{grade of "a is high", grade of "v is high"})$
 = the maximum of the following:
 $\min(\text{zero}(a), 0)$
 $\min(\text{low}(a), 0.25)$
 $\min(\text{medium}(a), 0.75)$
 $\min(\text{high}(a), 0)$
 $\min(\text{veryhigh}(a), 0)$
 = the maximum of the following:
 0
 $\min(\text{low}(a), 0.25)$
 $\min(\text{medium}(a), 0.75)$
 0
 $= \max(\min(\text{low}(a), 0.25), \min(\text{medium}(a), 0.75))$

Finally, because a is arbitrary, we get:

$$g(x) = \max(\min(\text{low}(x), 0.25), \min(\text{medium}(x), 0.75))$$

The above computing is also illustrated in Figure 1.17. (As it is obvious from the Figure, the operations of $\min(\text{low}(x), 0.25)$ and $\min(\text{medium}(x), 0.75)$ have the effect as cutting the membership function at a given level).

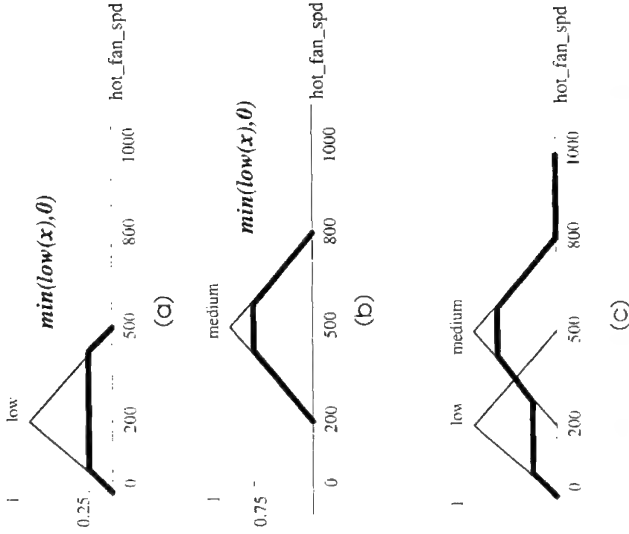


Figure 1.17 Output membership function obtained by aggregation
 (a) cutting the membership function $\text{low}(x)$ at level 0.25, (b) cutting the membership function at level 0.75, (c) obtaining the maximum (point-by-point) of the curves obtained in (a) and (b)

Similarly, suppose the grades of the output variable `cool_fan_spd` are:

Label	Truth Value
zero	0.75
low	0
medium	0
high	0

Then the corresponding output membership function is illustrated in Figure 1.18.

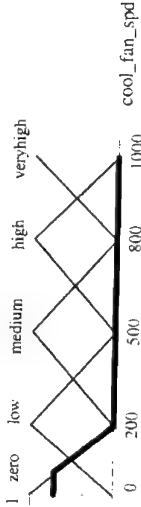


Figure 1.18 Output membership function of variable `cool_fan_spd`

After aggregation, defuzzification finds the center of the gravity of the fuzzy output. Figure 1.19 demonstrates defuzzification results for the two output variables.

```
hot_fan_spd = 440
cool_fan_spd = 65
```

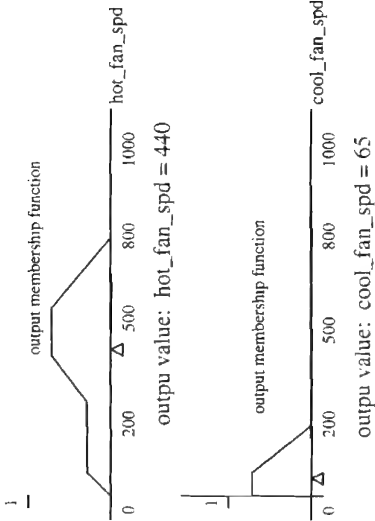


Figure 1.19 Defuzzification of output membership functions illustrated in Figures 1.17 and 1.18

In summary, in the case of Mamdani's method, the inference process consists of four steps:

- Fuzzification.
- Rule Evaluation.
- Aggregation.
- Defuzzification.

The main difference from the inference introduced before is the extra step aggregation. Figure 1.20 is the complete inference chart of the temperature controller.

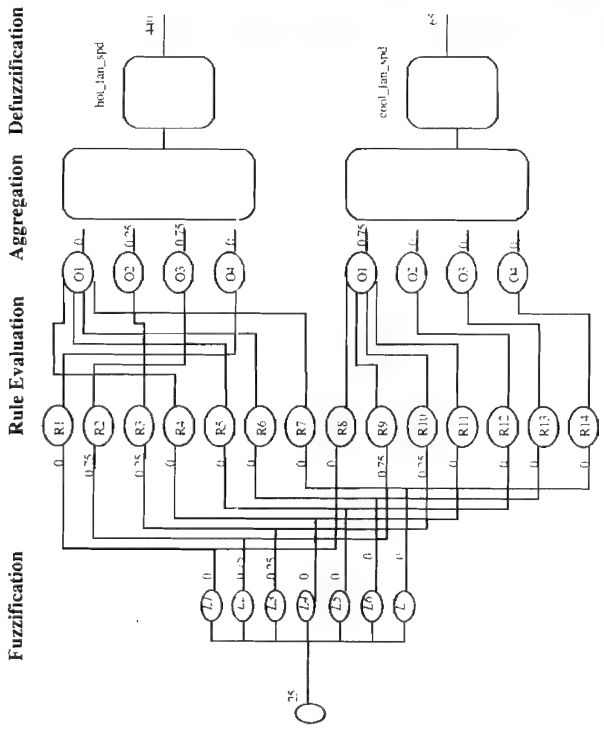


Figure 1.20 Inference chart for Mamdani's method

Singletons associated to output labels

We want to compare the two inference methods: the TVFI method and the Mamdani's method. However, we should mention again that the inference units to which the TVFI method applies and the inference units to which the

Mamdani's method applies are different: for the TVFI method, the output labels are associated with support values, while for the Mamdani's method, the output labels are associated with membership functions. The comparison seems less meaningful if the methods are applied to different units.

However, the difference is not important and it is easily eliminated by regarding each of the support values associated with the output labels as a singleton (the set with the value as its only member).

From now on, we will say that, (1) in an inference unit, the output labels are associated with membership functions (of fuzzy sets); and (2) if (and only if) all of the fuzzy sets are singletons, the TVFI method can be used for the inference unit.

It can be shown that Mamdani's method and the TVFI method produce the same result when applied to the same inference unit in which all output labels are associated with singletons. Therefore, the Mamdani's method is more general than the TVFI method (in the mathematics sense), or equivalently, the TVFI method can be regarded as a specialized case of Mamdani's method.

The advantage of the TVFI method is the absence of the tedious aggregation step (Figure 1.21). Actually, with the TVFI method, the fuzzy inference process is simply a process of generating and operating grades (or truth values) and letting them flow through an inference chart with AND and OR nodes, before the defuzzification step. The method is named from that process.

The advantage of TVFI also includes a simplification of defuzzification. For example, to obtain the centroid in Mamdani's method, numerical integration is necessary; while in the TVFI method, the integration is simplified to a weighted average.

In Fide, user can specify whether to use TVFI or Mamdani's method in an inference unit. However if ease of development and the efficiency of implementation are considerations, the TVFI method is recommended.

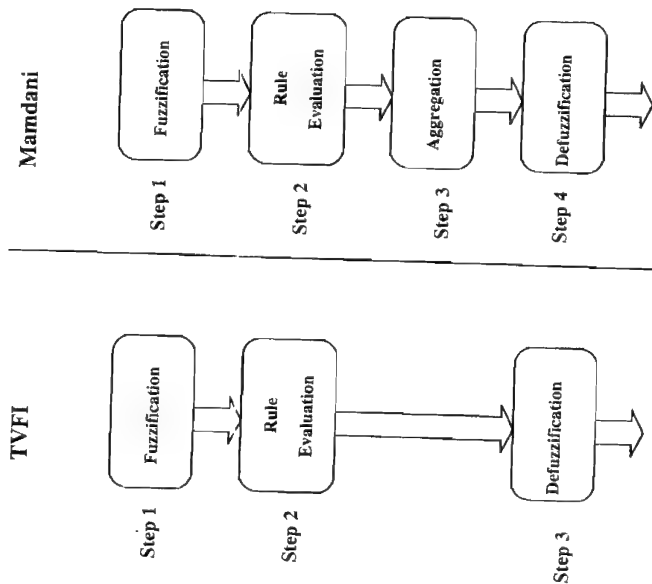


Figure 1.21 TVFI method Mamdani's method

Variations of fuzzy inference unit

Fuzzy supports a wide range of variations in fuzzy inference units. Some variations are important in applications.

More Logical Operators

In the above discussion, the logical AND operator is interpreted as maximization, and the logical OR as minimization. Besides these interpretations, we may select any two operations from the following table:

Symbol	Explanation	Definition
min	minimal operation	$\min(a,b)$
max	maximal operation	$\max(a,b)$
prod	product	ab
sum	probability sum	$a+b-ab$
binter	bounded intersection	$\max(a+b-1, 0)$
bunion	bounded union	$\min(a+b, 1)$

The selection of operations involves topics in the theory of fuzziness. Roughly speaking, if the input variables in a fuzzy inference unit are rather independent, prod and sum are preferred; but, if they tend to be correlated, the combination of min and max is better.

Fuzzy allows you to switch the operator easily, so as to allow you to compare the output of using different pairs of operators before you make a final determination.

More on Defuzzification

Defuzzification is the final step of producing an output value from the output membership functions. In the above discussion, we used (the abscissa of) the centroid as the defuzzified output value. This is equal to the weighted average (of the support values of the output labels), if the TVFI method is being used.

Fide provides more defuzzification operations called defuzzifiers as listed below (the simplified definition for the TVFI method is enclosed in parentheses):

- Centroid
The abscissa of the centroid
(The weighted average)
- Leftmost Maximizer
The leftmost point which has the maximal grade
(The leftmost support value with the maximal grade)
- Rightmost Maximizer
The rightmost point which has the maximal grade
(The rightmost support value with the maximal grade)
- Average Maximizer
The average of the leftmost and rightmost maximizers.

Figure 1.22 illustrates the result of each defuzzifiers, obtained from the output membership function in Figure 1.17.

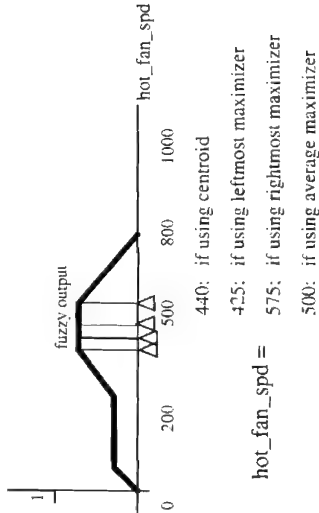


Figure 1.22 Defuzzification Methods

However, if the TVFI method is used, the output value can be determined for the support values of the labels and the grades obtained from the rule evaluation, as illustrated in the following table:

Label	Grade	Support Value
etc	0.25	200
low	0.75	500
medium	0	800
high	0	1000
very-high	0	

centroid = 425
leftmost maximizer = 500
rightmost maximizer = 500
average maximizer = 500

Fuzzy Systems

We discussed in the previous sections inference units. Let us move to a higher level and discuss fuzzy inference systems (fuzzy systems, for short).

A fuzzy system is a system in which (fuzzy) inference is used in one way or the other. Generally speaking, a fuzzy system consists of several units, among which some are inference units.

Let us consider the following example: to design a system simulating the function of the temperature controller.

The system should include several units: the controller, a thermodynamic model of the effect of the fans, and a comparing unit. The units are linked into a loop. The controller sends its output values of hot_fan_spd and cool_fan_spd to the thermodynamic model. The model computes the temperature at the next time cycle and sends the temperature to a comparing unit. It then checks the temperature against the desired target temperature and sends the difference to the controller. The data flow is illustrated Figure 1.23. Flow is from left to right except for the feedback loop.

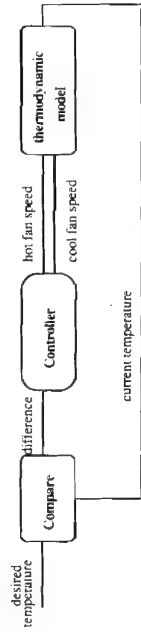


Figure 1.23 A Temperature Controller System consisting of a controller, a thermodynamic simulator and a comparing unit

The three units in the system, the controller, the simulator and the comparing unit, are of different type.

The controller is defined in terms of an inference unit. A fuzzy system usually contains one or more such units.

The comparing unit should have two input variables (the desired temperature and the current temperature) and one output variable (difference), and perform a simple mathematics operation -- subtraction. Such a unit is said to be an operation unit.

The simulator, including the thermodynamic computations and possibly graphic display functions as well as keyboard command interpreters (for setting mode, for example) is supposed to have been developed elsewhere and to be accessible through a specified interface. The details of the interface, including the input and output variables and the name of the callable function in the simulator program, should be defined as a special unit. Such a unit is said to be an execution unit.

To summarize, the units in the system are specified below:

- Units:
- Controller (CNT): inference unit
 - input variables: temperature_diff
 - output variables: hot_fan_spd, cool_fan_spd
 - rules: (specified elsewhere)

- Simulator(SIM): execution unit
- input variables: hot_fan_s, cool_fan_s
 - output variable: temperature;
 - to execute "xxxx" (a file name),
 - call the function "...." (a function name)
- Comparing unit(CMP): operation unit
- input variables: desired, current
 - output variable: difference
 - operation: difference = current - desired

The three units are illustrated in Figure 1.24. A small square is attached to each unit at its corner, where a letter 'I', 'E' or 'O' is marked to show the unit is an inference unit, an execution unit or an operation unit respectively. It is also mentioned that only the variables are shown because we are at the system level and concerned only with the interface between the units. The Figures are the internal views of the interface.

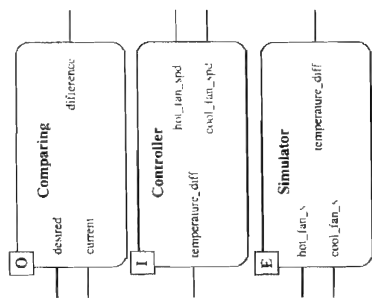


Figure 1.24 The internal views of the interface of the units

Now we can draw the system chart, showing how the data (the values of the variables) flow between the units (Figure 1.25). In the system chart, the units are views from outside, so that the variables of the units are not shown. They can be determined by referring to Figure 1.24. The lines in the system chart indicate the flowing direction of the data (from left to right, as a convention). Two variables are shown explicitly at the left and the right: `desired_temp` (for desired temperature) and `current_temp` (for current temperature). They are called system variables. The occurrences of the same variable `current_temp` at both sides indicates that the system is a closed loop, and the output value of the variable `current-temp` is fed back to the input as the input value in the next cycle. On the other hand, the variable `desired_temp` occurs only at the left, and therefore it stays constant at a fixed value when the system is initialized.

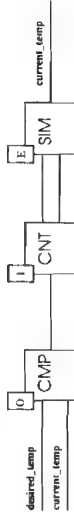


Figure 1.25 The system chart showing how the units and the system variables are linked together

To summarize, the variables and the links in the system are specified as below:

Variables:

`desired_temp`
`current_temp`

Links:

`send desired_temp to desired of CMP`
`send current_temp to current of CMP`
`send difference of CMP to temperature of CNT`
`send hot_fan_spd of CNT to hot_fan_s of SIM`
`send cool_fan_spd of CNT to cool_fan_s of SIM`
`send temperature of SIM to current_temp`

After the introduction to fuzzy logic in the first chapter, we now show how to apply that fundamental knowledge using Fide. This chapter gives a tour of Fide and highlights the tools of Fide and how they work. To illustrate the tools, a file called `FANRULE1.FIL` and another one called `FANRULE2.FIL` are used as examples. These files are found in the `FANS` subdirectory and the path to them is `C:\FIDE\EXAMPLES\FANS`. Both files are Fide inference units that adjust the speed of a hot air fan and a cool air fan to control the temperature. The hot air fan blows out air that is always 110 degrees while the cool air fan blows out air that is always 0 degrees. The temperature in a room is varied by changing the speed at which each fan turns, from an 100% to completely off so that no air blows out of the fan. Whatever the initial temperature might be, the hot and cool air fans come on accordingly to bring and keep the temperature at the desired temperature. This desired temperature can be set by the user or a default value of 70 degrees is used.

Writing and Compiling Source Code

Let's open the example file `FANRULE1` and take a look at it. To open `FANRULE1`, choose `Open under File` on the Fide main menu. Now make sure Directory reads `C:\FIDE\EXAMPLES\FANS`. If the directory listed is different than what you want, use the subdirectory names and two dots inside brackets in the Directories box to move to the proper subdirectory. Once you are in the proper subdirectory, highlight the name `FANRULE1.FIL` as shown in Figure 2.1

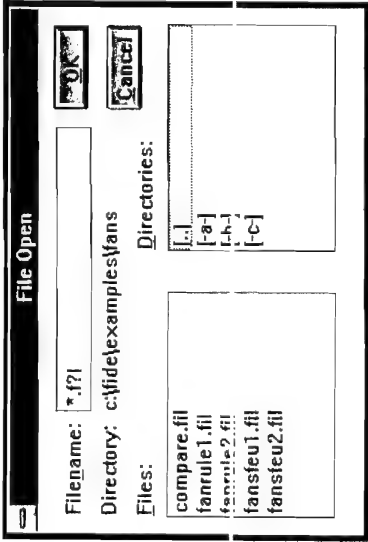


Figure 2.1 File open dialog box

Click on the O.K. button and the source code for the FANRULE1.FIL file is opened up onto the screen. A listing of this source code is shown below for reference as the source code is discussed.

```
$flu for a Temperature Control system
$this is only an example..The rules may be improved
$Date created: April 23, 1992 $Date last modified: April 28, 1992
$Filename: FANRULE1

%% In this example, the input variable is
% temperature_diff=desired temperature - current temperature
% the output variable is
% hot_fan_spd= speed of the hot air fan
% cold_fan_spd= speed of the cold air fan

Shneider: It is a flu.
flu tvfi (min max) *8;

$An input variable and its membership functions
invar temperature_diff "degree": -110(1)110 (
    NegativeBigDiff (8-110,1,8-50,1,8-30,0),
    NegativeMediumDiff (8-50,0,8-30,1,8-10,0),
    NegativeSmallDiff (8-30,0,8-10,1,8-0,0),
    ZeroDifference (8-10,0,8-0,1,810,0),
    ZeroDifference
```

```
PositiveSmallDiff (80,0,810,1,830,0),
PositiveMediumDiff (810,0,830,1,850,0),
PositiveBigDiff (830,0,850,1,8110,1));

outvar hot_fan_spd "rpm": 0 (5) 1000 * (
    zero =0,
    low =200,
    medium =500,
    high =800,
    veryhigh=1000);

$Another output variable and its membership functions
outvar cool_fan_spd "rpm": 0 (5) 1000 * (
    zero =0,
    low =200,
    medium =500,
    high =800,
    veryhigh=1000
);
```

Rules describe the relations between input variable and output variables
\$Format: if input is label then output is label
if temperature_diff is PositiveBigDiff then hot_fan_spd is high;
if temperature_diff is PositiveBigDiff then cool_fan_spd is zero;
if temperature_diff is PositiveMediumDiff then hot_fan_spd is medium;
if temperature_diff is PositiveMediumDiff then cool_fan_spd is zero;
if temperature_diff is PositiveSmallDiff then hot_fan_spd is low;
if temperature_diff is PositiveSmallDiff then cool_fan_spd is zero;
if temperature_diff is ZeroDifference then hot_fan_spd is zero;
if temperature_diff is ZeroDifference then cool_fan_spd is zero;
if temperature_diff is NegativeSmallDiff then hot_fan_spd is zero;
if temperature_diff is NegativeSmallDiff then cool_fan_spd is low;
if temperature_diff is NegativeMediumDiff then hot_fan_spd is zero;
if temperature_diff is NegativeMediumDiff then cool_fan_spd is medium;
if temperature_diff is NegativeBigDiff then hot_fan_spd is zero;
if temperature_diff is NegativeBigDiff then cool_fan_spd is high;
end

This chapter does not give a rigorous discussion of the syntax for the source code file. A description of how to use the fuzzy inference language (FIL) and a detailed description of its syntax can be found in the *Fide Reference Manual*. Let it suffice to say at this time that all Fide inference units (FIU) are written with FIL.

The first thing to notice about FIL is that the dollar sign (\$) indicates the line it appears in is a comment. Thus, the first eleven lines in the listing for FANRULE1 are comments. Below the comments is the *header* for this file. A header is used to indicate the type of fuzzification and the fuzzy inference method. It is necessary for each FIU and explained in detail in the *Fide*

Reference Manual. Next we come to the input variable clause. Each input variable clause is denoted in FANRULE1 by the word **invar** which is a reserved word. FANRULE1 has one input variable - temperature_diff. This variable is the difference between the actual temperature being measured in a room, for instance, and the desired temperature in that room. FANRULE1 also has two output variables, one each for the speed of the hot fan and the speed of the cool fan. The name of the output variables are hot_fan_spd and cool_fan_spd. These are denoted by the reserved word **outvar**. Shown below is a block diagram of the control system.

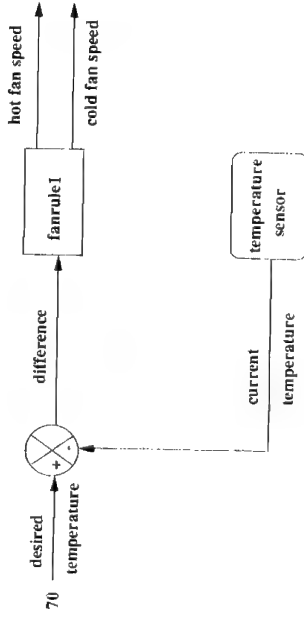


Figure 2.2 Fanrule1 system diagram

When designing a Fide inference unit, the first step is always to decide on the input and output variables. All other steps in the design of the Fide inference unit will depend upon the variables picked as inputs to it and the outputs from it. In the diagram above of our temperature control system, it is easy to determine the input and output variables of the Fide inference unit. Since we want the temperature in the room to be a certain temperature, we simply measure the difference in the actual and desired temperatures and input the difference into the control unit (Fide inference unit). Then, based on this difference, we change the speed of one or both fans to raise or lower the

temperature in the room. Picking the variables are not always as simple as this because a system is often more complex, but once a rough description of what needs to be controlled is outlined, the variables are easy to spot.

Referring back to the source code listing, you will notice the word **degree** inside parenthesis after the declaration of **temperature_diff** as an input variable. This declares **degree** to be the unit of measurement for the variable **temperature_diff**. Following **degree** is a colon and then a group of numbers, **this group of numbers defines the range of the variable to be from 110 to 110**. The one inside the parenthesis defines the increment over the variables range for each cycle or step of the control system. The seven names listed under the input variable, and shown in Figure 2.3 below, are called **labels**.

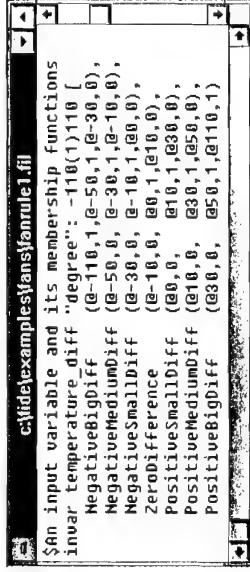


Figure 2.3 Labels for input variable **temperature_diff**

All the labels define the *membership function* for a variable. That is to say, the membership function spans the entire range of the variable and the labels make up the membership function. The first label defined is **NegativeBigDiff** which stands for negative big difference. This name is very descriptive and we immediately know what it means when we read the name. This label covers the range of the input variable for the area when the measured temperature is much larger than the desired temperature. All label names should be descriptive like **NegativeBigDiff** so the source code is easy to use.

After NegativeBigDiff we find some numbers inside of parenthesis which define the shape of each label. Each pair of numbers defines a shape of the label with the first number representing the temperature coordinate and the second number the truth value coordinate. Thus, the numbers can be read as, "at -110, the truth value is 1", and "at -50, the truth value is 1" and "at -30, the truth value is 0". The three pairs of points when taken together represents a membership function shape. If you are uncertain about exactly what values to give your labels don't be concerned - Fide provides powerful debugging and tuning tools to help you adjust the membership function for optimum performance. The next section of the text will show how to edit the membership functions with the graphical editor.

The next part of the source code is the output variable clause. As we noted before, the output variable `hot_fan_spd` is indicated with the word **outvar**. Also, note that the output variable name immediately identifies what is being controlled, the speed of the hot fan. The range for the output variable `hot_fan_spd` is from 0 to 1000 rpm's and for each cycle of the control system the output variable increments by 5. The labels for the output variable are defined differently from the labels of the input variables because the *TVFI* method is used. In the TVFI method, all output labels are singletons which results in reduced time for defuzzification and less memory usage. For more information on this method please refer to Chapter 1.

For general applications, the membership functions usually give the best performance when they overlap each other at about 50% grade. The diagram below shows the membership function for the input variable `temperature_diff`. Notice how each label intersects around the 0.5 mark.

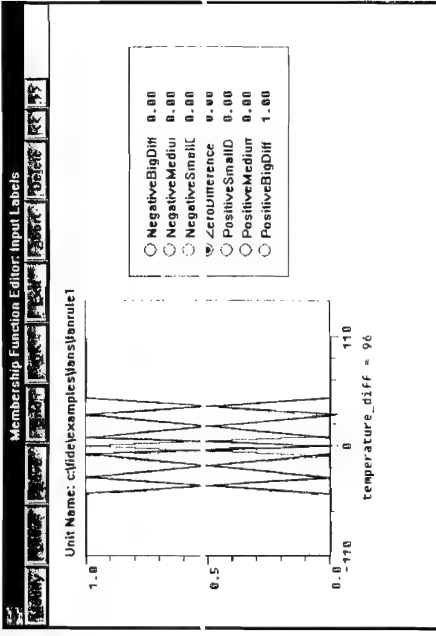


Figure 2-4 Membership function for input variable `temperature_diff`

Now all that is left to complete our Fide inference unit is defining the *rules*. The rules simply describe the relationship between the input variables and the output variables. The syntax of FIL is very easy to use and follows intuition. Thus, if the difference in the measured and desired temperature is a large negative difference (like NegativeBigDiff) then the measured temperature must be very hot. Remember the difference in temperatures is calculated by desired temperature - current temperature. So if the current temperature is very hot then naturally you want to turn on the cool fan speed full blast. This intuitive statement gives rise to the last rule shown in the listing above which reads:

if `temperature_diff` is NegativeBigDiff then `cool_fan_spd` is high.

All of the rules are written in this manner - they follow the intuition that a person applies in solving a problem.

With the variables, their membership functions, and rules defined we can compile the source code. First, make sure your file is saved and then click on the Compile option from the main menu of Fide. You will see a drop down menu appear with two options on it - Source file and Input file. Choose Source file to compile FANRULE1.FIL. You see some messages from Fide indicating that the file is being compiled and when the compiler has finished you see the message in Figure 2.5 below displayed on your screen



Figure 2.5 Successful compilation message

Lets modify the FANRULE1 source to see how errors are reported in case a mistake is made while writing source code. Lets erase the close parenthesis on NegativeBigDiff, the first label of the first input variable. Now save the file to disk and re-compile it. The screen shown below appears with a message window below the source code listing error messages. The first error message is highlighted and indicates the line and column on which the error occurred for easy reference. Usually, the first error message is the most important one because many of the other error messages are residual effects of the real error. The other error messages disappear when the error causing the first error message is fixed. You can also access the position of the error by double clicking on the error message and the cursor jumps into the source code to the position of the error. Replace the close parenthesis and re-save FANRULE1.FIL to disk. Compile it now and there are no error messages.

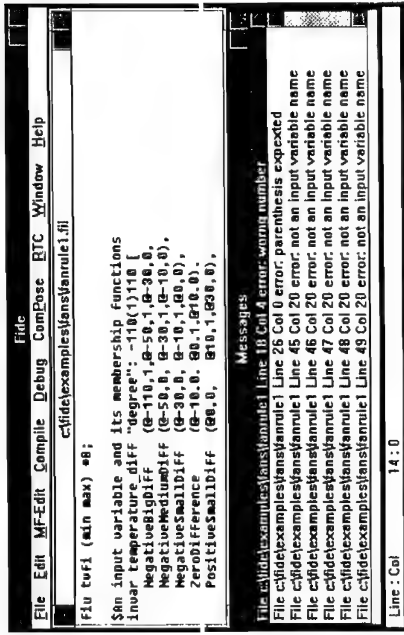


Figure 2.6 Message window with errors

Debugging With The Fide Tracer

To debug your Fide inference unit means to optimize its performance. This is done by modifying the rules and membership functions that you defined in the source code. One of the tools Fide provides to make debugging easier is the Tracer. To get into the tracer, click on the Debug option on the Fide main menu and click on Run Debugger in the drop down menu. From there you see the main window for the Debugger with two menu items. One is called Target and the other is called Tools. Click on Tools and a drop down menu appears with three choices on it, Tracer, Analyzer and Simulator. Click on the Tracer option and then you see the screen depicted below.

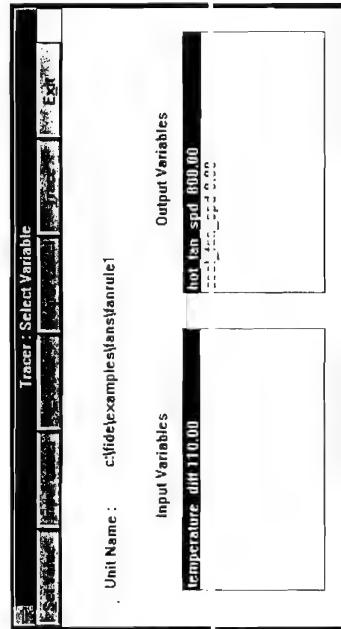


Figure 2.7 Tracer main window

From here we can display the input and output membership functions. To see the membership function of temperature_diff make sure the input variable temperature_diff is highlighted as seen above and then click on input label. The diagram shown below is seen on your screen.

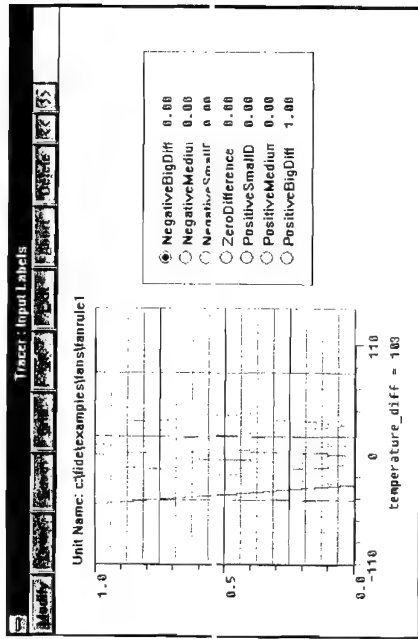


Figure 2.8 Membership function for input variable temperature_diff

Notice in the box to the right of the graph that the circle for NegativeBigDiff is colored in. This indicates that the label highlighted on the screen always corresponds to the label whose circle is colored in. From here, press the Modify button on the upper-left side of the Tracer: Input Labels screen to see the Label Dialog screen shown below.

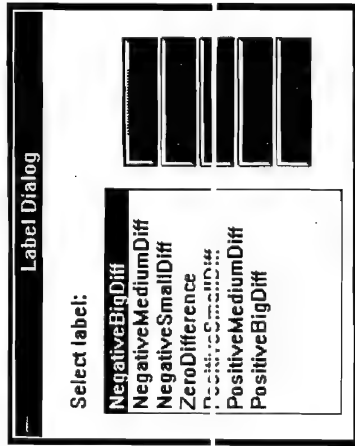


Figure 2.9 Label dialog window

This dialog box enables you to modify the labels for purposes of experimenting with different shapes to observe the effects on the output values. If the different shapes give better results, then the source code for your FIU can be modified to permanently reflect the temporary changes made here in the graphical editor.

To use this feature, highlight whichever label you want to modify and then choose the method of modification. There are three available, Operation, Draw, and Polygon. The MF-Edit option on the Fide main menu supports the same modify functions - please refer to the *Fide User's Guide* on how to use them. The difference between the Tracer and the MF-Edit option is that the tracer does not make changes to the source code (the changes are in memory only for your current session) but the MF-Edit option changes the values of the membership functions in the source code as explained in the *User's Guide*.

The Tracer is used to follow the path of an output data value back to the rules in the source code that fired to contribute to the final defuzzified value. To begin this process, highlight one of the output variables listed under Output Variables on the Tracer: Select Variable window shown in Figure 2.7 and note the value listed next to the output variable you have highlighted. Then go to the menu bar and click on Trace. This opens the window shown below.

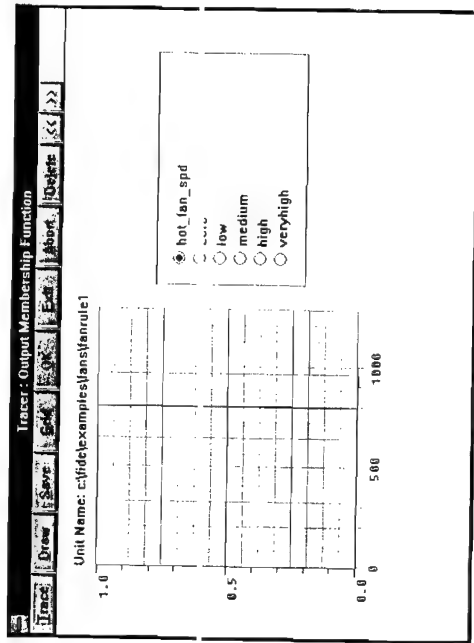


Figure 2.10 Output labels

This window is the graphical representation of the singleton output labels. It shows the degree to which each output label contributes to the final output value. Now go to the upper left button on this window and click on Trace. The Tracer: Select Output Label window appears in Figure 2.11. From this window highlight a label from the Labels box to trace.

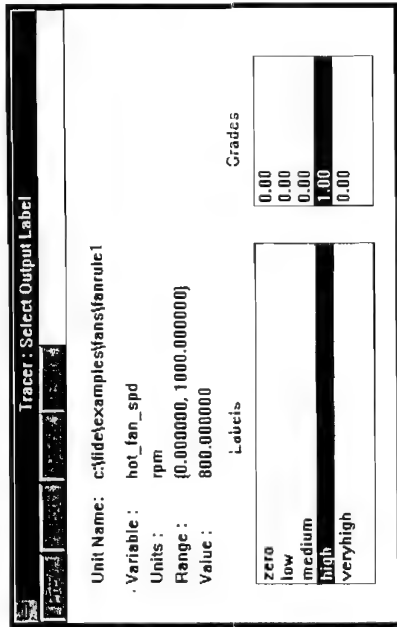


Figure 2.11 Select Output Label window

Click on **high** to highlight it as shown in Figure 2.11 above. Clicking on the Prev or Next buttons highlights the previous or next variables respectively. With the proper label highlighted, click on Trace and the window shown below appears on the screen.

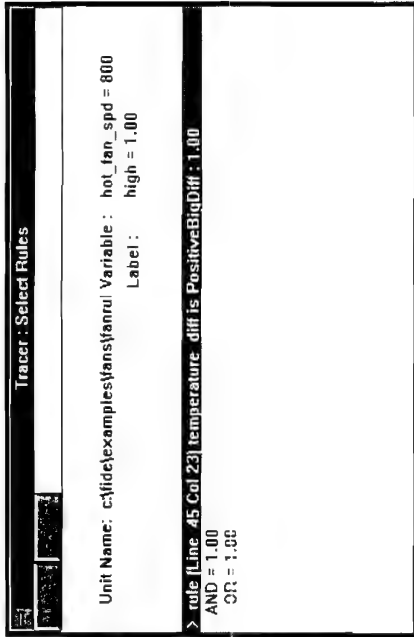


Figure 2.12 Select Rules window

This window shows the antecedents from rules that contributed to the final output value. For the particular output value we are tracing, 800, only one rule contributed to the output value, thus, there is only one antecedent displayed. Normally, many antecedents would be displayed and one would be highlighted. Now select Trace again from the menu bar at top. This takes you to the message window, and the source code as shown in the diagram below.

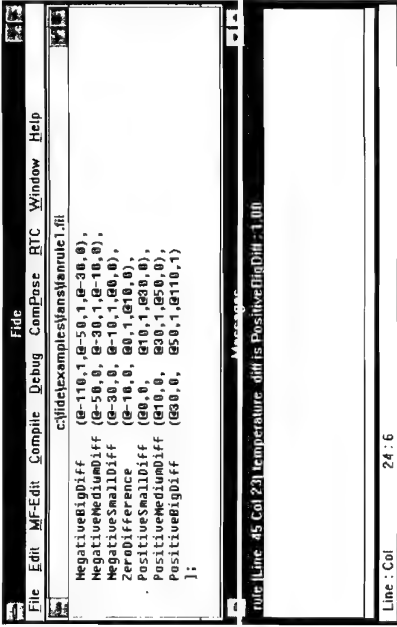


Figure 2.13 Source code traced from output value

Now simply double-click on the highlighted antecedent in the message window and the cursor jumps into the source where the antecedent appears. Now you can modify the source code to meet your needs.

Debugging With The Fide Analyzer

To demonstrate how the Fide Analyzer operates, we are going to have to make our example just a tad more complicated by adding another input variable. The reason why will become evident as we examine the Analyzer tool. The source code we will follow for the analyzer is called FANRULE2 and the block diagram is shown below.

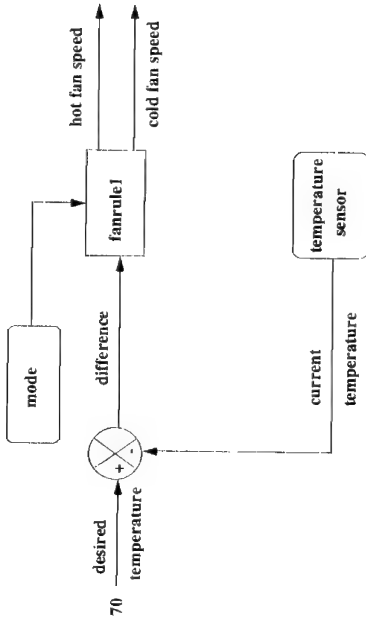


Figure 2.14 Fanrule2 system diagram

There are two major differences in the source code of FANRULE2.FIL, from FANRULE1.FIL. There is another input variable and its associated labels declared, and there are more rules to deal with using a second input variable. To use the Analyzer first open the FANRULE2 source code if you have not already done so and compile it. Then choose the Debugger option from the main menu and click on Run Debugger which displays the Debugger main window on your screen. As before, click on the Tools option of the debugger main window. But this time choose the Analyzer selection instead of Tracer. When you choose the Analyzer option, your screen will display the image shown in Figure 2.15 below. This is the main window for the Analyzer tool.

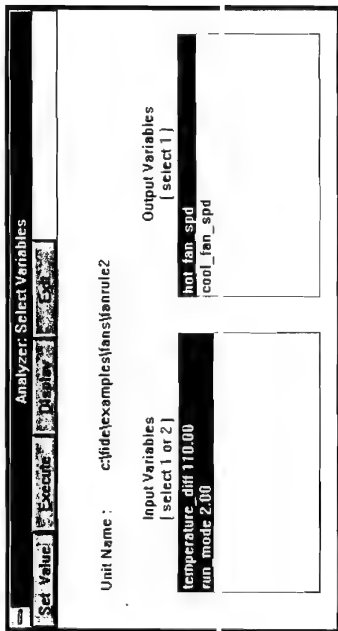


Figure 2.15 Analyzer main window

Notice that we now have two input variables listed under the Input Variables box. Run_mode is the additional input variable that is added to FANRULE2. From this screen we create the data points to generate a three-dimensional surface, a contour view of the surface and a side view of the surface. The three dimensional surface view has to have two input variables and one output variable highlighted for a total of three variables to generate data since it is in three dimensions. This is the reason we have to add another variable to FANRULE1. In Figure 2.15 above, temperature_diff and hot_fan_spd is highlighted so we need to click on run_mode so there are three variables selected for the surface view. The reason we are highlighting hot_fan_spd is because we want to see the control surface for this variable. If it was desired to view the control surface for cool_fan_spd then we would highlight the cool_fan_spd variable. With three variables selected, click on the Execute button and the following dialog box (Figure 2.16) appears.

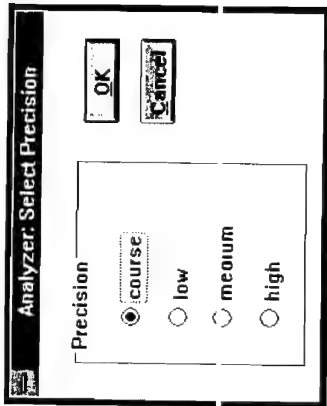


Figure 2.16 Select precision window

The Select Precision dialog box allows varying degrees of precision in the data points generated to be selected. Course takes about twenty seconds for data points to be generated and high takes about twenty minutes. Just click in the circle to select which precision category you desire then click on O.K. and the data points are generated. While the data points are being generated, you will see an hourglass which indicates that Fide is working. As soon as the hourglass is gone and you see the pointer again, click on the Display button of the Analyzer: Select Variables window. The next window that appears on your screen has three options in the menu bar. Click on the Views option to reveal the drop down menu as shown in Figure 2.17 below.

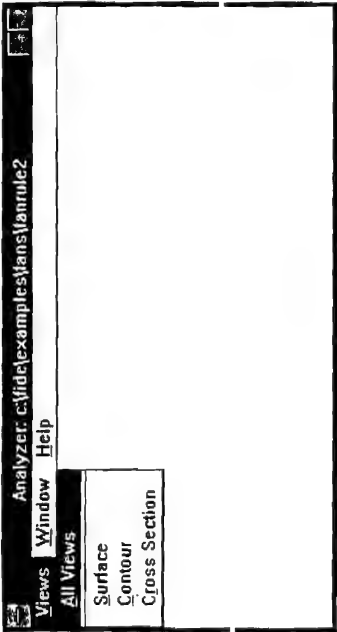


Figure 2.17 Choose desired view

Click on the All Views option which is already highlighted and Fide will draw the Cross Section view first, then the Contour view and finally the Surface view. You cannot make any view the active one until all three are drawn so it is best to let all three views be drawn first, and then rearrange them to your liking. If you like, each view can be drawn one at a time by choosing one of the three views on the drop down menu listed below. All Views shown in Figure 2.17. The following picture depicts the views cascaded after all three have finished being drawn.

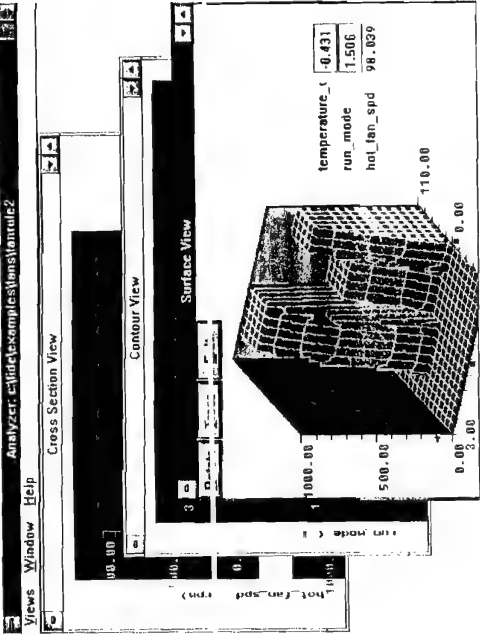


Figure 2.18 All three views shown in cascade

Let's use the surface view for debugging. The vertical axis of the surface represents the output variable while the two input variables are represented by the horizontal axes. Each variable represented on the surface is listed on the right side of the screen along with the value for that variable. The red arrows on the horizontal axes can be dragged back-and-forth to vary the value of that variable. As the input variable values change, the output variable value changes accordingly. This allows you to see the exact output value for any combination of the input values. If an anomaly has occurred at any point on the surface, you can trace back to the source code that generated this point. Then you can examine and modify the membership functions and rules to correct the anomaly. In addition, any point on the contour view can be traced into the source code as well. To go back into the source code click on the Trace

button. This begins the Tracer tool that was described earlier in the text. For a detailed description of how the Analyzer works please refer to the *Fide User's Guide*.

Using The Fide Simulator

Another tool provided with Fide to help debug your FIU is the Simulator. To use the Simulator, an input file has to be prepared that contains a set of test data. The Simulator is used to test the FIU against any set of test data desired, when the simulation is done, a graph of the input and output variables is displayed on a Cartesian coordinate graph. This allows you to look at the response of the Fide inference unit and double-check the analyzer results. Let's test our FIU, FANRULE2, against a set of test data. First open the file entitled FANRULE2.FDL up onto the screen. This file is opened like any other file, simply click on open from the Fide main menu and highlight FANRULE2 when you see it and then click on O.K. - make sure you are in the FANS sub directory. As you can see, the input file is very easy to prepare. Simply list the input variables and their range then the data you want. The variables are separated by commas and so are the numbers. The last input variable before the numbers begin has a semicolon following it instead of a comma. Each set of numbers is separated with a semicolon as well. This file gets compiled with the Input file option under Compile. You get the same type of messages when you compile an input file as you get when you compile a source file. Once the input file is compiled then you can go to the Simulator to test the FIU with your test data. Click on Debug from the Fide main menu and click on Run Debugger. This opens up the Debugger main window from which you click on the Tools option. Then from the drop down menu choose the Simulator option. This opens the Simulator main menu onto the screen which looks like the picture shown in Figure 2.19.

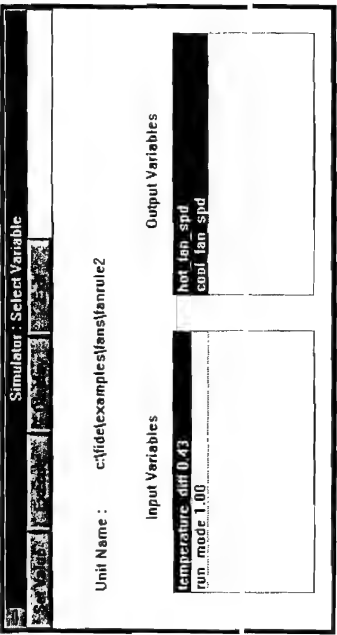


Figure 2.19 Simulator main menu window

We first need to click on run_mode and set its value to 0.1 or 2 which represent energy saver, normal or fast mode. Click on Set Value when run_mode is highlighted and a new window pops up with run_modes current value highlighted. Replace the highlighted value with 0.1 or 2 by simply typing the number. The highlighted value is automatically replaced with the value you are typing. Now go to the Enter button and click on it so run_mode is set to a proper value. The other variables shown in Figure 2.19 are used in the simulation. We will not select run_mode because its range is defined to be 0 to 3 which is much smaller than the other variables that have a range from -110 to 110. Thus, it will not show up on the graph.

Click on the two output variables and the input variable temperature_diff so they are highlighted as shown in Figure 2.19. With the proper variables highlighted, click on the Execute button. You will see the hourglass icon while data points are generated. When the hourglass turns into the pointer again, click on the Display button. Figure 2.20 will be displayed on the screen. This is the plot of the variables highlighted in the Simulator: Select Variables Window.

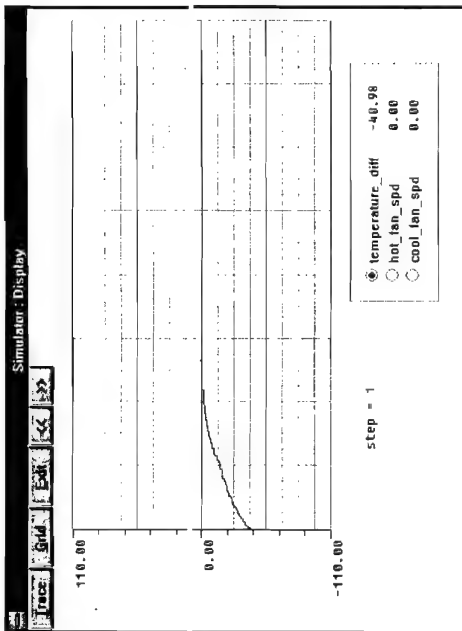


Figure 2.20 Simulator graph

On the vertical axis of the plot is the temperature of the variables defined in the source code FANRULE2. Click on the circles next to the variable names in the box on the bottom right side of the screen to highlight each variables graph. The horizontal axis is the step that the Fide inference unit has executed. If you move the pointer into the graph then a red vertical line appears and the step is indicated at the bottom of the graph. The Tracer can be utilized from this screen to correct any undesirable behavior of the variables output by tracing back to the source code as described before. Click on the Trace button to initiate this process.

Real-time Code Generator

The Real-time Code Generator is accessed through the RTC option on the main menu of the Fide window. It generates the assembly code for the MC6805, MC68HC05 and MC68HC11 Motorola MCUs. The assembly code gets stored in files with the extension of .asm and with the same name. We will generate 68HC05 assembly code for the FANRULE2 FU. Make sure FANRULE2.FIL is open and compiled then click on the RTC option and a drop down menu appears with MCU in it. Now click on this and a menu appears as shown in Figure 2.21 below with the three Motorola MCU choices.

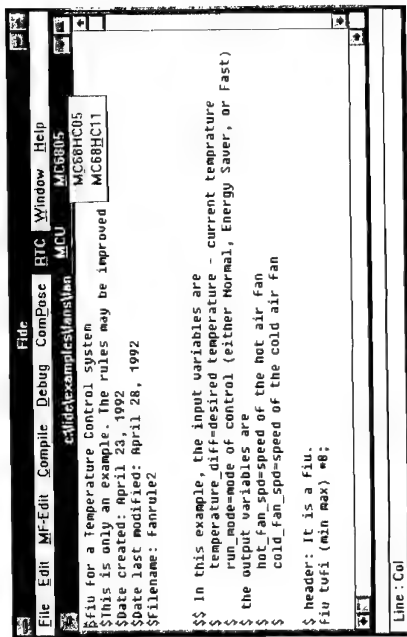


Figure 2.21 RTC option on Fide main menu

Click on MC68HC05 and then you will get a message that the Real Time code is being generated. Then the following message appears indicating that the assembly code was generated successfully.



Figure 2.22 Successful conversion message

Now you can open up the FANRULE2.asm file to look at the assembly code. Click on the File option on the main menu bar of Fide and choose open from the drop down menu. Enter *.asm in the Filename box at the top of the File Open window so your screen looks like Figure 2.23.

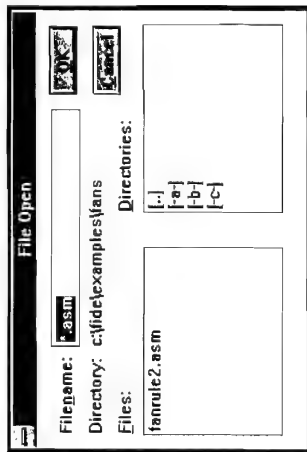


Figure 2.23 File open window

You can click on O.K. from here and observe the assembly code for the FANRULE2.FIU. This code can be assembled by the IASM 05 assembler. Then the object code can be downloaded onto the target hardware for execution.

Chapter 3 Developing A Fuzzy Inference System

The development of a Fide inference unit and the tools to code, test, debug and tune that FIU were covered in the last chapter. This chapter covers how a FIU is integrated with non-fuzzy inference units to create a system that can simulate how the FIU operates in a real world application.

Fide Composer and Fide Library

The Fide Composer uses three types of modules to develop a system. They are the FIU (Fide inference unit) FEU (Fide Execution Unit) and the FOU (Fide operation unit). The three module types are covered in detail in the *Fide Reference Manual*. Lets begin our discussion of the Composer by taking a look at the system diagram of the temperature control system that FANRULE2 controls. Make sure all of the FEU, FOU and FIUs have been compiled by the Fide compiler before a system diagram is opened. Otherwise, error messages appear on your screen. To see the diagram. Click on the Composer option on the main menu of Fide and click on Switch to Composer in the drop down menu. This brings up the main menu of the Composer shown in Figure 3.1.

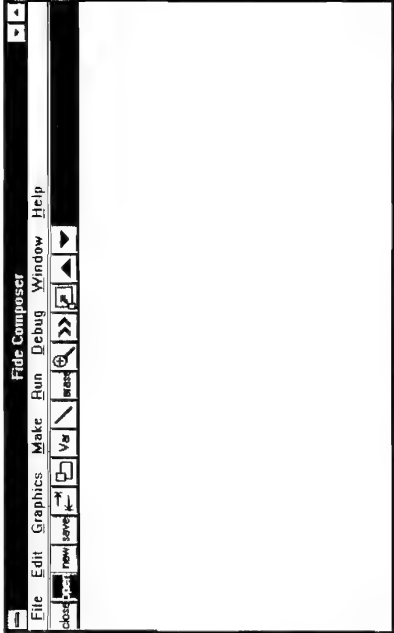


Figure 3.1 Composer main window

Now click on the second button, open, on the Composer tool bar. This opens the screen shown below. (You may have to change directories to get to the one shown in Figure 3.2 below.)

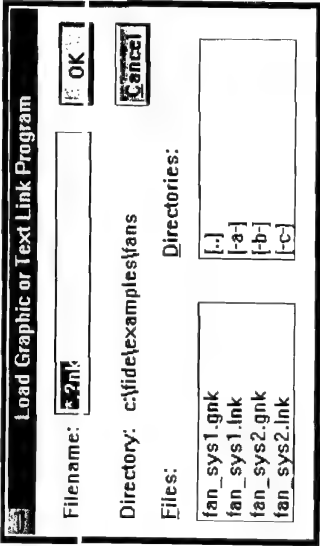


Figure 3.2 Load program window

Click on fan_sys2.gnk and then O.K. and the system diagram for the temperature control system appears. The diagram is shown below.

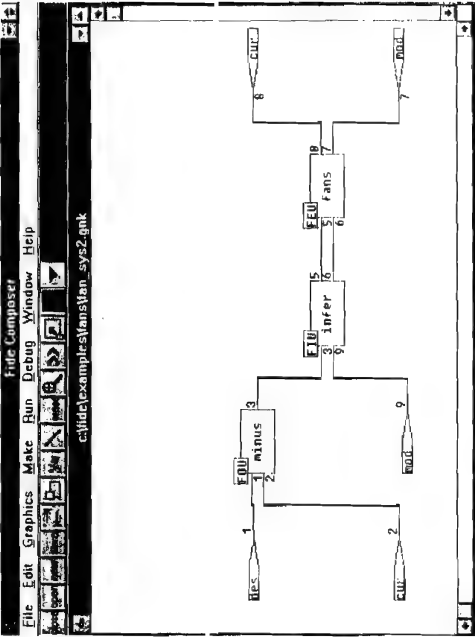


Figure 3.3 Fan_sys 2 graphical diagram

This diagram represents a closed loop temperature control system that is controlled by FANRULE2. The source code we designed is contained in the module named infer (in red letters). It also has FIU in blue letters on the upper left corner of the block to indicate what type of module it is. It is connected to an FOU named fans (also in red letters). This module simulates the real world action and enables the software simulation. The FOU named minus (in red letters again) calculates temperature_diff by subtracting the current temperature from the desired temperature. The other objects in the diagram are system variables. Cur is the current temperature, mod is the run mode of the fans (high, normal or energy saver) and des is the desired temperature.

The Fide library allows the object code of FANRULE2 to perform the fuzzy logic calculations. The Fide library is compatible with the Borland C and

Turbo C compilers. When the system depicted above is compiled the header file is automatically included in the compiled code. So to perform software simulations you only have to build the diagram and compile it. We will cover how to build the system diagrams in the next section. The Apronix Run Time Library is covered in detail in the *Fide Reference Manual*.

Using the graphics editor

To illustrate how to use the Composer graphics editor, we will build a part of the fan_sys.gnk diagram. To begin, open up a new, clean diagram slate. Do this by clicking on the File...New Graphics option on the Composer main menu as shown in Figure 3.4.

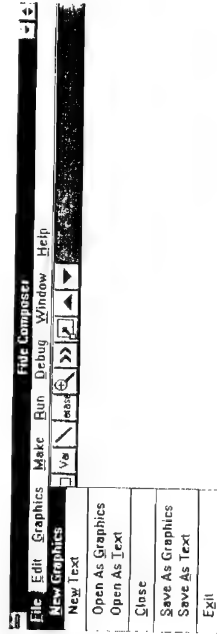


Figure 3.4 File drop-down menu

You get a new diagram with the title of Untitled.gnk to begin building your graphical system. Now click on the fifth button of the Composer tool bar and then double-click anywhere in the Untitled.gnk window and you will see a window like Figure 3.5.

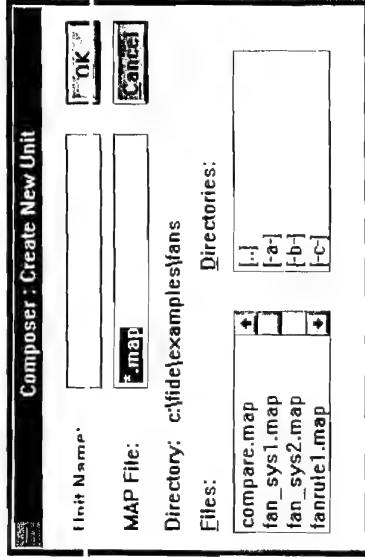


Figure 3.5 Create a New Unit window

The second box with the highlighted name, *.map, in it is a file that is generated by the Fide compiler for every module. The MAP file contains information needed by the Composer so that it can integrate the modules into a system. You don't need to worry about it more than making sure that every module you use in a system is compiled by the Fide compiler so the MAP file is present. For more information on MAP files please consult the *Fide Reference Manual*.

Now, choose the FANRULE2.map file from the Files box by clicking on it and the name will appear in the MAP File box. If you don't see FANRULE2.map then scroll the Files box with the arrows until it appears. Then click on the Unit Name box and type in a name. Let's make the name infer, to be consistent

with the existing temperature control system. Your screen should look like Figure 3.6 now.

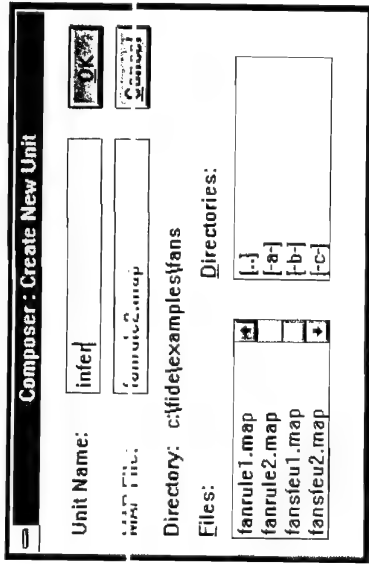


Figure 3.6 Enter Unit name and MAP file name

From here click on O.K. and the first module appears on the Untitled.gnk window as shown in Figure 3.7.

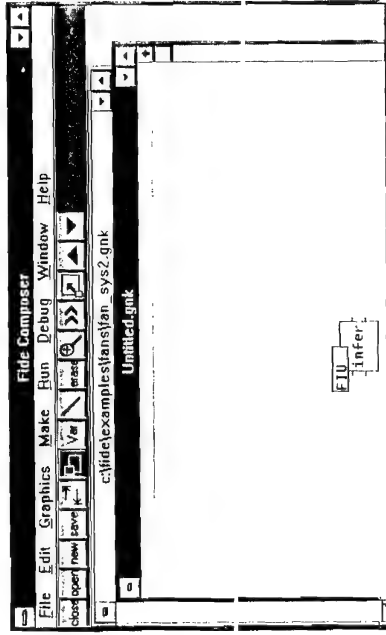


Figure 3.7 First unit appears on system diagram

Now we have to add the other components to the system. We will add the FEC module which is found in the Files box as FANRULE2 and call it fans. Follow the same procedure as above so that your screen looks like the Figure 3.8 and then press on O.K. and you now have two modules on your diagram.

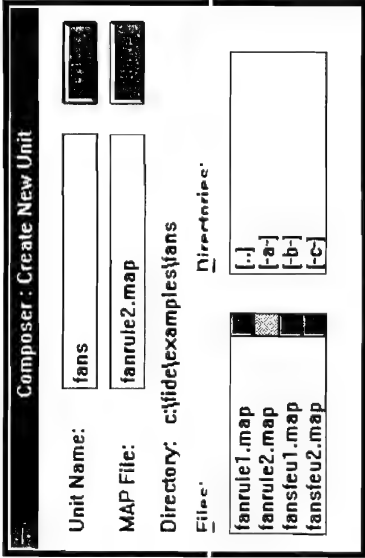


Figure 3.8 Enter name and MAP file name for second unit

Lets add the FOU to our system now. Follow the same procedure to put it on the diagram as for the FEU and FIU. The FOU is in the Files box with the name compare.map and lets make the unit name minus. With the FOU in the diagram, we now need to add a system variable. To add this variable click on the seventh button of the Composer tool bar first. Then go to the untitled window and double click. You will see the Composer: Create New Variable screen. Since this is an input variable, click in the circle beside Input in the Type of Variable box. Now click in the Variable Name box and type in des for desired so your screen looks like Figure 3.9. This is the desired temperature input variable.

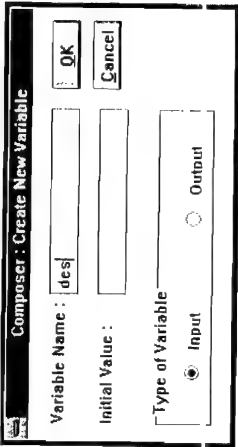


Figure 3.9 Create a new system variable on diagram

Click on O.K. and the system variable appears on your Untitled.gnk system diagram as shown in Figure 3.10 below.

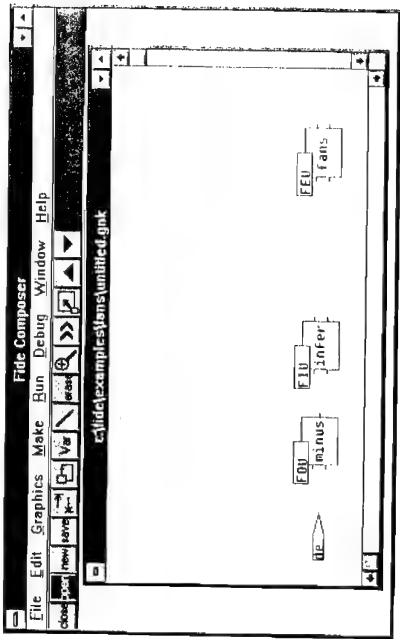


Figure 3.10 System variable appears in diagram

Now add another input system variable and name it cur so your diagram looks like Figure 3.11 below.

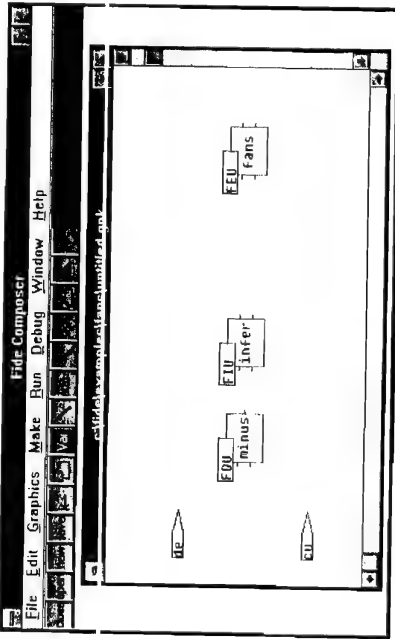


Figure 3.11 Second system variable in diagram

Now we have the basics on how to put modules into the system diagram so we need to connect the modules with data flow paths. This is done by clicking on the eighth button of the Composer tool bar. Move the mouse pointer into the untitled.gnk diagram and it turns into a pencil. With the edge of the pencil, double click on the end of the des system input variable module. You will see a screen as shown in Figure 3.12 pop up. Make sure to highlight des so the Composer knows which variable to work with - it will not default to des even though it is the only variable shown.

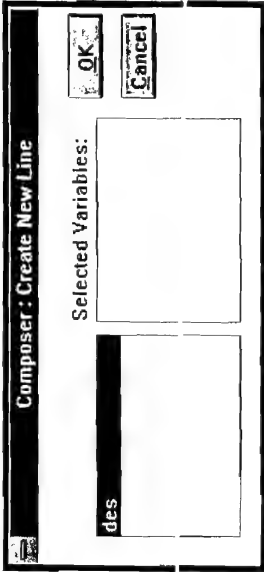


Figure 3.12 Indicate source unit to create a dataflow path

Click on O.K. and then double click on the first node of the module called minus and the screen depicted below pops up. Highlight the name desired in the Inputs box to tell the Composer that you want to connect the input system variable des to the module minus.

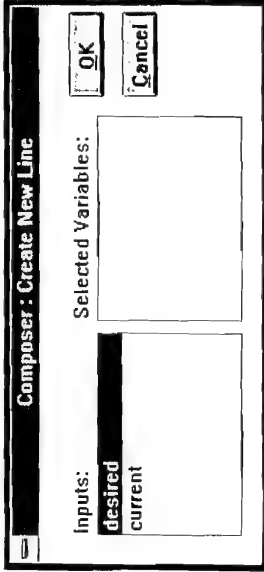


Figure 3.13 Indicate sink for dataflow path

Now click O.K. and a data flow path, numbered 1, connects the system input variable des to the module minus. This is shown in Figure 3.14.

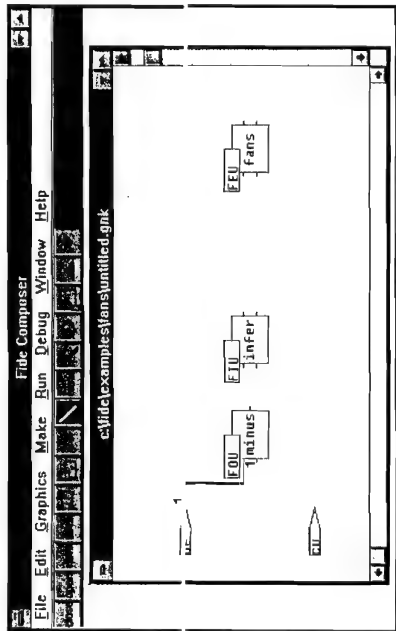


Figure 3.14 Dataflow path appears in system diagram

Using the same procedure, connect the system input variable cur to the module minus and remember to highlight the variable cur and current at each Composer: Create New Line pop up window. Now we need to connect the module minus to the module infer. Click on the eighth button of the Composer tool bar again and double click on the minus node on the right side of the minus node. The following window will appear.

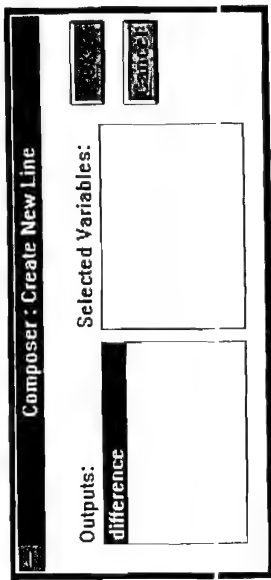


Figure 3.15 Creating another dataflow path

Highlight the variable difference and click on O.K. Difference is the result of the calculation: (desired temperature - current temperature). Now double click on the first node of the infer module and highlight the temperature_diff variable. Click on O.K. and the third line is drawn on your diagram as shown in Figure 3.16. Creating a system diagram with the Fide Composer is as simple as that! Practice connecting the rest of the diagram by creating data flow paths between the modules infer and fans using the procedures above.

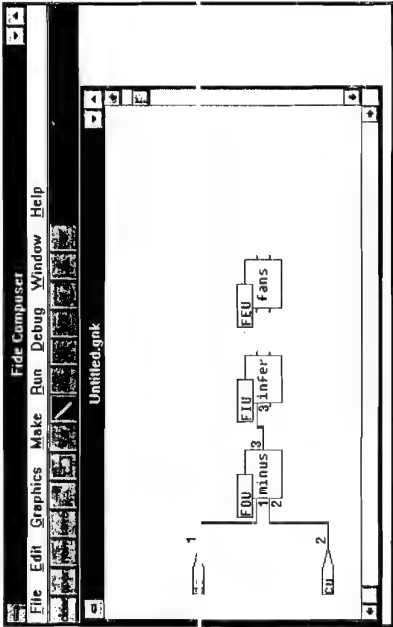


Figure 3.16 Second dataflow path appears

You may have noticed the option *Save as Text* on the drop down menu of *File* from the main menu of Fide as shown highlighted in Figure 3.17.



Figure 3.17 *Save As Text* option

This option will save a graphical diagram as text and stores it with the same name but a *.lnk* extension instead of *.gnk*. In addition, a text file of a system can be saved as a graphical diagram. You can also open a graphical file as text and vice-versa. This is provided in case a user desires to build their system with the text editor instead of the graphical editor.

Making an executable file

Making an executable file is simple to do with the Fide Composer because it does all the work for you. To make the executable file a Borland C or Turbo C compiler is needed and the executable file will run on a DOS environment. Open the *fan_sys.gnk* or *fan_sys.lnk* file onto the screen. Then choose the *Make* option from the Composer main menu. Click the *Make* for Execution

option highlighted in Figure 3.18 and the Composer generates `fan_sys2.exe`. You will see a black DOS window appear with some text indicating the C compiler is linking the files of the system. When you see the DOS prompt in this window the compiler is done and you can close this window by clicking on the button in the upper left corner of the DOS window. Then select the Close option from the drop down menu.

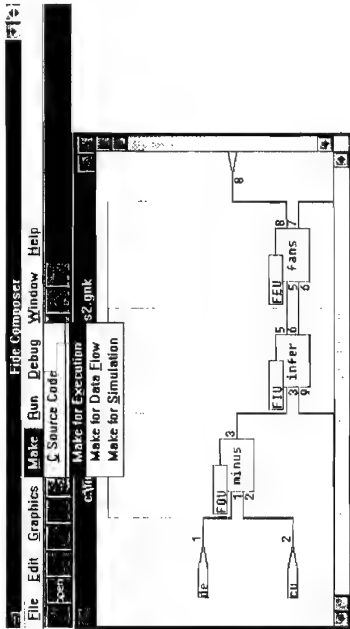


Figure 3.18 Make drop-down menu

Now click on the Run option from the Composer main menu bar and you will see the drop down menu shown in Figure 3.19.

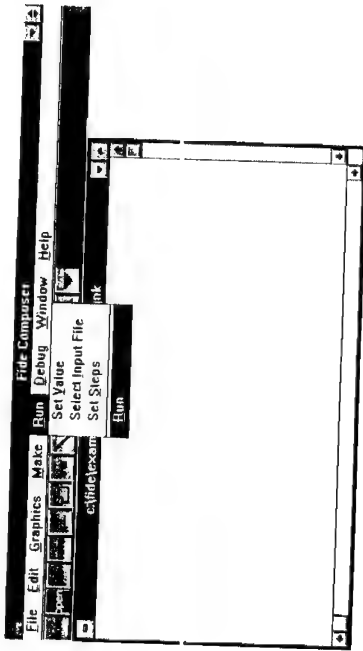


Figure 3.19 Run option

This is the time to put the desired temperature into the system if you do not want the default value of 70 degrees. Click on the Set Value option and the Composer Set Value window lists all of the input variables for the system. Highlight desired and click on the Set Value button. Another window appears that has the present value for desired highlighted. Type the value you want desired to have and click the Enter button. This will give desired the value that you want. Now the software simulation can be run.

Click on Run that is highlighted in the figure and the software simulation will run. There is a title screen from which you can press any key to start the simulation. Follow the directions on the screen to change the run mode and vary the initial temperature. You may have noticed on the drop down menu for Make shown in Figure 3.18 that there are four options. The C Source Code

option generates a file in uncompiled C that can be ported to another DOS machine if the user desires. The C source code can be compiled on any DOS machine that has a Borland C or Turbo C compiler. The Make for Data Flow and Make for Simulation options generate files that are slightly different than the Make for Execution option because these files are used for debugging purposes. These two options are explained in the next two sections.

Using The Data Flow Viewer

Generating a data flow view executable file is exactly the same as generating a regular executable file except the Make for Data Flow option is clicked on from the Make drop down menu. You will see the C Compiler Message window again with linking messages from the C compiler. When you see the DOS prompt then close the black DOS window. Now click on Run.Run and let the software simulation run for a little bit. While the simulation runs, a data file is generated that stores data values to display for the Display Data Flow view. The Display DataFlow view is an option on the drop down menu of Debug from the main menu of the Composer. The Display DataFlow option is shown highlighted in Figure 3.20.



Figure 3.20 Debug drop-down menu

Click on this option and you will see a choice of a Stepwise or Continuous display Data Flow. Lets click on the Stepwise option to display the data one step at a time. After clicking on Stepwise, you will see numbers printed on the screen just past the des and cur system input variable symbols. Move the mouse pointer onto the diagram screen and it will turn into a double right arrow. Double click your mouse and numbers are printed just after the next modules in the system, the minus module and the mod system variable symbols. Each time you double click the mouse, data values progress through the system one step further. This will continue until the end of the data file is reached.

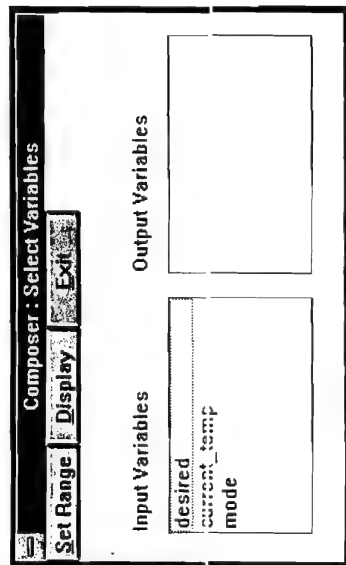


Figure 3.23 Select Variables window

The default range for all the variables that get displayed on the Composer simulator graph is from 0 to 255. Since the range of many variables may be much different than that, much smaller for instance, the variables will not show up on the graph. In our example, the graphs for each variable show up but we will use the Set Range capability of the simulator so we know how it works. Highlight the variable whose range you wish to change (select desired) and then click the Set Range button. The screen shown in Figure 3.24 appears on your screen.

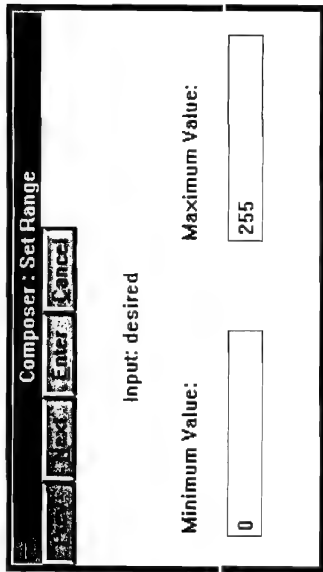


Figure 3.24 Set Range window

Notice the default range is from 0 to 255. Highlight the value 0 in the Minimum Value box and enter -110. Then highlight 255 in the Maximum value box and enter 110 so your screen looks as pictured below.

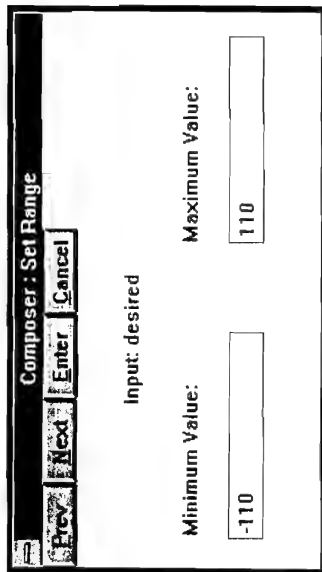


Figure 3.25 Set Range window with the proper values entered

Now click on the Enter button and this range is used for the variable desired. Now lets change the range of the other two variables to match the range of desired. Highlight current temp and click on the Set Range button. Enter -110 and 110 into the Minimum Value and the Maximum Value boxes as before. This time, instead of clicking on the Enter button, click on the Next button. The next variable, mode, appears on the Composer: Select Range window next to Input:. This way you can change the range on each variable by opening up the Composer: Select Range window one time instead of for each variable. When you have set the range for all the variables, click on Cancel in the Select Variable window and the range is set to -110 to 110 for all the variables.

Lets click on all three variables listed under the Input Variables box to select them for display on the simulation graph. Click on display and you get a graph with a plot of the three variables highlighted on the Composer: Select Variables window. The graph will look similar to Figure 3.26 below except there will be grid lines on the graph. This image was captured after the Grid button was clicked to remove the grid lines.

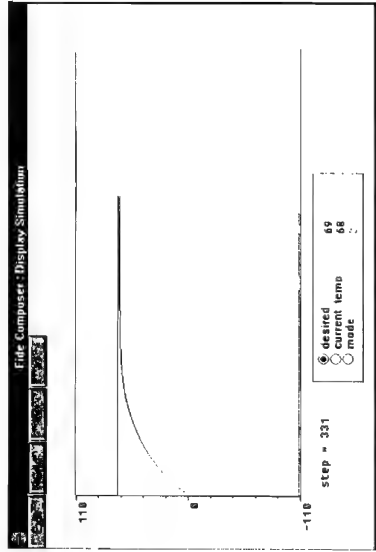


Figure 3.26 Graph of the Simulation

Click on the circles next to the variable names in the box on the bottom right side of the screen to highlight each variables graph. The horizontal axis is the step that the Fide inference unit has executed. If you move the pointer into the graph then a red vertical line appears and the step is indicated at the bottom of the graph. Before we leave the Composer, one more item on the Debug drop down menu should be mentioned. This item is the Trace option and is highlighted in the menu shown in Figure 3.27.

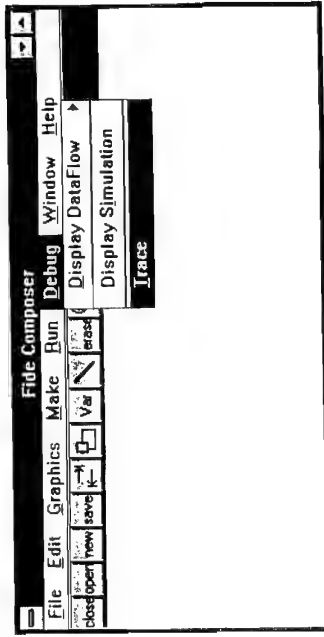


Figure 3.27 Trace option is available on Composer also

Choosing this options allows any of the Fide inference units in a system to be traced. When this option is highlighted, the mouse pointer turns into an icon that looks like a box with an arrow when moved onto the system diagram screen. Double click on any Fide inference unit and the main window of the Fide Tracer is displayed as shown below.

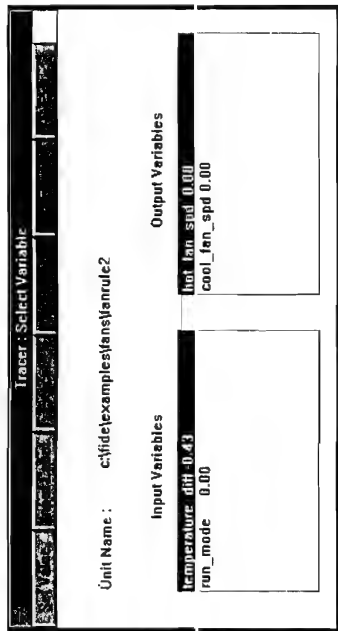


Figure 3.28 Main window of Tracer

From this point Tracing the FIU is exactly the same as described before.

Introduction

What Is Fide?

The word Fide is an acronym for Fuzzy Inference Development Environment. Fide the software product provides the tools to create, maintain, and utilize fuzzy inference units.

At the heart of Fide is a compiler. The compiler accepts English like rule syntax and concise notation for variable and membership function definitions, making it easy for you to specify any fuzzy logic application. Moreover, Fide's compiler is modeled after traditional language compilers, such as C, using freeform text as input. This familiar environment provides tremendous flexibility for you as a developer, who can use any text editor, including Fide's own text editor, to create and modify rule source in the style, in the sequence, and with embedded in context comments most appropriate for the application you are responsible for.

As an integral part of the development environment, Fide also provides a full complement of visualization and debugging tools. You can see your fuzzy inference input/output transfer function in a variety of graphical views, ranging from 3 dimensional surface, to topographic layers, to cross sectional planes. You can visually pinpoint a coordinate of interest and automatically trace back to the rule source for the originating statement. Furthermore, you can simulate in graphical format a real time run of the input and output variables as they vary over a user definable duration of time. You can get a visual feel of the performance of your fuzzy inference system in terms of responsiveness, stability, and accuracy all within your development environment.

Fide also supports the conversion of compiler object code into specific MCU chip format. Among others, Fide supports the Motorola MC6805, MC68HC05 and MC68HC11 chips. Code is generated into standard assembly language source code, which can be assembled by a standard MCU assembler.

Once individual Fide Inference Units have been developed, you can link multiple units together (cascaded or in parallel) with external units into an application system. Fide provides a tool called Composer, which allows you to

Introduction

represent Fide Inference Units and other units symbolically in order to connect them together using a simple line drawing tool.

Fide is a complete fuzzy logic development environment. With it you can create fuzzy applications faster, more intuitively, and expect results that are reliable and robust. Along the way you will come to understand your application more clearly and more deeply as you use straightforward analysis and debugging tools to fine tune and verify your product.

Hardware and Software Requirements

The following minimum configuration is required to run Fide.

- MS-DOS 3.3 or later
- MS Windows 3.0 or later
- A personal computer with an 80286 processor (or higher) and 2 megabytes (MB) or more of memory.
- A hard disk with 5 megabytes of free disk space, and at least one floppy disk drive.
- A VGA color monitor supported by Windows.
- A mouse supported by Windows. Fide requires a mouse. Fide supports drawing functions that can only be utilized through mouse cursor movement, and not through equivalent keyboard commands.
- A parallel port (may be the same port used by a printer) to which the Fide hardware key (Hard Key for short) is attached.
- One of the following C compilers: Turbo C 2.0 or later, Borland C 2.0, or Microsoft C 6.0.

Fide Package

You should find in your Fide package the following items. If any item is missing or defective on arrival, contact your dealer or distributor immediately for repair or replacement of the item.

- Distribution Media (3.5, 2.5" 1.2MB or 3.5" 1.44MB diskettes)

- Fide Hardware Key and serial number
- Fide Quick Start Guide
- Fide User's Manual (This manual)
- Fide Language Reference Manual

Using The Manuals

The manuals included with your Fide Product are intended to serve two functions: (1) assist you in becoming proficient in using Fide as quickly and effectively as possible, and (2) provide an authoritative source of reference information on the usage conventions and command syntax requirements of Fide. Each manual is described briefly below.

Fide Quick Start Guide

This manual gives you the basics to begin performing productive work on Fide. It includes installation instructions and the steps needed to launch Fide on your computer.

Fide User's Manual

The User's Manual is grouped into two major parts: (1) a tutorial on using Fide and Fuzzy Logic, and (2) a comprehensive description of Fide functions available through Fide's menuing system.

Fide Tutorial

The Fide tutorial provides a step by step tour of the Fide software product using a consistent and instructive application throughout. Starting from a rule source file (included on your distribution diskettes) representing the example application, the tutorial takes you through the process of compiling, debugging, and simulating a Fide fuzzy inference unit. Further lessons include compiling

input source for Fide simulation, and launching the Fide Composer to link and debug an entire system comprised of individual Fide inference and operation units and an external execution unit. If you are a first time user of Fide or are unfamiliar with Fuzzy Technology in general, going through the steps of this tutorial will give you a hands-on introduction of these topics and is highly recommended.

Fide User's Guide

The Fide User's Guide section describes every feature of Fide from a user point of view. Each menu item, each interactive graphic, and each informational display is explained. It contains the "how-to" instructions for getting the most effective results out of Fide.

This manual describes how to generate C language code from your Fide compiler output and how to link these C routines to your standalone C application. Since Fide generated C code is ANSI compatible, Fide code is not restricted to only supported controller chips but can be ported to any platform supporting an ANSI C compiler.

Fide Reference Manual

The Fide Reference Manual contains technical specifications, including language syntax and data structures, for several man-machine and machine interface aspects of Fide.

FIL Reference

FIL stands for Fide Inference Language. This manual presents and explains the language syntax of FIL. FIL is used to define Fide Inference Units (FIU), Fide Operation Units (FOU), and Fide Execution Units (FEU).

FCL Reference

FCL stands for Fide Composer Language. This manual presents and explains FCL syntax.

Aptronix Run Time Library Reference

This manual describes the Aptronix Run Time Library (RTL) included with your Fide package. The Run Time Library consist of C language object routines which are linked into your application before the program is executed. Each function is explained in terms of its syntax and is use through one or more instructive examples.

Typesfaces Used In Fide Manuals

To facilitate the use of Fide, menu and dialog box command examples are used extensively in several of the manuals included in the Fide package, particularly the Tutorial and User's Guide. Several typographic conventions are adopted in Fide manuals to make clear the intended meaning of words in the context of a user interactive environment. They are as follows:

Type style	Used for
Monospaced type	Anything that you must type exactly as it appears. For example, if you are asked to type <code>Get.Fide</code> you type all the monospaced characters exactly as they are printed in this guide.
<i>Italics</i>	Signals a new term. An explanation immediately follows the italicized term.
bold	Place holder for information you must provide. For example, if you are asked to type filename , you would type the actual name for a file instead of the word shown in bold print. This typeface is also used to indicate a reserve word.
ALL CAPITALS	Directory names, filenames, and acronyms

**SMALL
CAPITALS**

The names of keys on your keyboard. For example, CTRL, ESC, or HOME. A plus sign (+) between key names means to hold down the first key while you press the second key. For example, Press ALT+F means to hold down the ALT key and press the F key. Then release both keys.

**Initial
Capitals**

Menu items, command names, and dialog-box names and options. For example, File menu, Save command, or Course option.

How to Contact Apronix

Apronix is committed to providing the highest level of customer support possible. There are 3 means by which users of Fide can contact Apronix for technical support. They are:

BBS

Apronix maintains an internal BBS (Bulletin Board System), called FuzzyNet, where interested parties can download sample files, technical updates, and other information on Fide related issues through use of a modem. Users can also use the BBS to upload source files or other pertinent information to Apronix for special attention. FuzzyNet also supports E-Mail messages for on-line communications with Fide users. No special setup, other than a modem and basic communications software, is required. To access FuzzyNet call (408) 428-1883.

FAX

Users can FAX in a technical request to Apronix at (408) 428-1884 with a return FAX number or phone number, and Apronix will respond to the request either with a suggested solution or a time frame for a solution.

Voice

Apronix can be contacted directly via telephone at (408) 428-1888 between the hours of 9:00am and 5:00pm Pacific Standard Time. If the question is about Fide, please have your product serial number available and call from a phone near the computer on which you are running Fide.

Installing Fide

Setup

Before installing Fide on your computer, it is highly recommended that you make a backup copy of the original Fide diskette set (3 disks) and store it in safe place.

Fide runs under Microsoft Windows 3.0 or 3.1. Installation occurs under Windows control. To install Fide you must have 5 megabytes of free disk space available on your hard disk. Insert Disk 1 of your distribution media set into drive A: of your computer. Key in the following command:

```
at> win setup
```

You will be prompted to confirm or enter in the name of the default directory to which Fide files will be copied. Unless you change the default, Fide files will be stored in the directory Fide. This prompt is shown below in Figure Intro-1.

Follow the on-screen instructions for inserting diskettes. Once all files have been installed successfully, a completion dialog box appears as shown in Figure Intro-2.

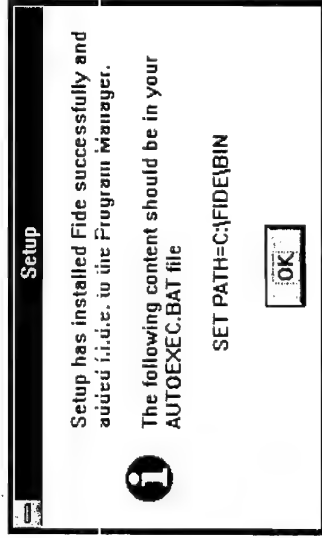


Figure Intro-2 Setup Successful Dialog Box

Fide setup does *not* automatically modify your AUTOEXEC.BAT file to include C:\FIDE\BIN in your path statement. You must perform this modification yourself. This feature is intended so that your system configuration is not changed without your knowledge.

Hardware Key

Fide will not run unless you have installed the hardware key included in your Fide package. Plug the hardware key into your computer's (IBM compatible PC) parallel port 1 (LPT-1). If you have a device (e.g., printer) already attached to the parallel port, detach its connector and reattach it on top of your Fide hardware key. In other words, your Fide hardware key is inserted between parallel port 1 and any pre-existing parallel device.

Fide Basics

Fide runs under Microsoft Windows 3.x. It is assumed that the reader is familiar with the basics of MS Windows operations. If not, please read your Microsoft Windows User's Guide before proceeding. Fide conforms to standard Windows usage for mouse and keyboard interaction with Windows elements such as menu commands, push buttons, scroll bars, list boxes, radio buttons, and edit boxes. This Fide User's Manual also conforms to Windows terminology such as "select" and "choose". Refer to your Windows User's Guide for instructions on how to use Windows elements and the definition of Windows terms.

Starting Fide

Once Fide has been successfully installed, you can start Fide from either DOS or Windows as follows:

From DOS

> cd fide

From Windows

Bring up the Fide group and double click on the Fide icon as shown in Figure Intro-3 below.

For menu commands

menu1/menu2/menu3 ...

where *menu1* is the top level menu item
menu2 is a menu item under *menu1*
menu3 is a menu item under *menu2*
and so on

Chapter 1 Creating Source Files

Fide provides a complete text editor for creating and maintaining source files. The operating principle is to select or create a text object and then act on that object. The object could be a text file or a block of characters within the file. Fide provides the tools to enter and manipulate text.

The Fide menu system provides access to all the tools available for designing and developing fuzzy inference units and systems. Menu commands can be initiated using the mouse and keyboard cursor commands (point and click and arrow keys respectively), or you can use the ALT-letter accelerator keys to directly initiate a command. The letter in the ALT-letter sequence is the underlined letter of the command name in the menu.

The following sections describe menu groups of the Fide menu that pertain to source file creation and management. Other menu groups under Fide are covered in later chapters.

File Menu Commands

The Fide file system is central to Fide coordinating its activities throughout the development process. And the central file used by Fide is the Rule Source File. All other files are derived from or used by the Rule Source File. How these files are used and how they inter-relate are the subjects explained in detail in remaining chapters of this User's Guide.

Files are managed through the tools described in this section. The File menu is shown in Figure 1-1 below:

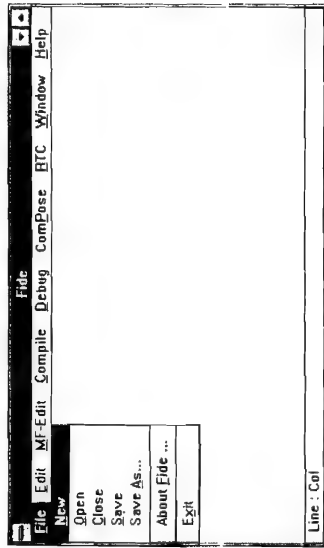


Figure 1-1 Fide Menu

New

The New command brings up an untitled blank text edit window into which you may immediately begin typing source statements. Descriptions of text editor functions are given in the Editor Menu section below.

Open

The Open command brings up a dialog box in which you are asked to open an existing file for Fide processing. The dialog box is shown in Figure 1-2.

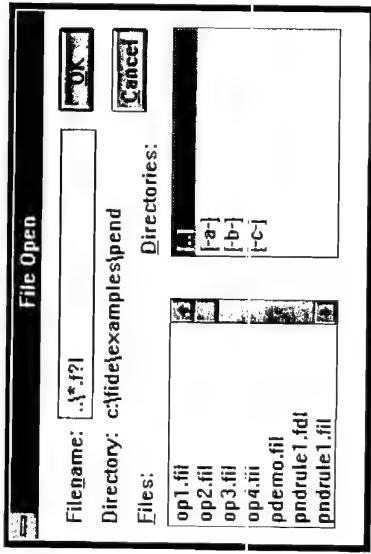


Figure 1-2 Fide/File/Open Dialog Box

You may enter a file path name in the edit box at the top of the dialog box, or you may choose a file name from the file list box at the left side of the dialog. To change directories, choose from the names shown in the list box to the right. You may also enter a file "filter" or "mask" in the file name edit box. Use the DOS conventions of * and ? as wild card characters (* for a variable number of characters and ? for a single character) in your filter name entry. The file list box will display only those files matching your filter.

You can open more than one file into the Fide Window. The file operated on is the file in the topmost window. This is the active or current window, which can be distinguished by its highlighted caption bar. The ability to open and manage more than one window within a single host window is known as Multiple Document Interface (MDI).

Close

The Close command closes the currently active window (title highlighted) without exiting Fide. If there are multiple windows open, the next most recently active window now becomes the active window once again. If you close a window for which changes have not been saved, a warning dialog box appears as shown in Figure 1-3.



Figure 1-3 File Changed Dialog Box

Save

The Save command writes to disk the file you are currently editing. The current edit window is the active window, which has its caption (title) bar highlighted.

Caution: You must save your current edit changes to disk before Fide will recognize it for subsequent processing, such as compilations. The Fide compiler takes as input the file which is stored on disk, not its file image in memory.

Save As

The Save As command performs the same function as the Save command, except that the file is written to a new disk file rather than overwriting an existing one. A dialog box comes up in which you are prompted for a file name. This dialog box is shown in Figure 1-4.

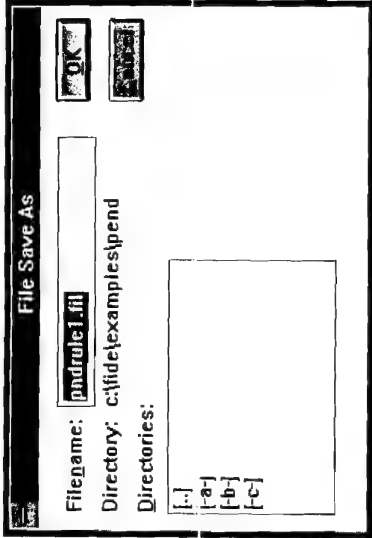


Figure 1-4 Fide/File/Save As Dialog Box

You can change the directory in which to store your named file by choosing a directory name from the list box shown in the dialog box.

About Fide

The About Fide command brings up a window containing information about the Fide Program. The version number and copyright notice are among items displayed.

Exit

The Exit command closes the Fide application in Windows. This command is equivalent to the Close command available from the Control menu of the Fide window.

If you have made changes to any file for which you have not already saved to disk, this command brings up a dialog box indicating the file has changed and asking you if you want to save the file before exiting. This dialog box is shown in Figure 1-3.

Edit Menu Commands

The Edit menu provides commands to manipulate text in the active edit window. The Edit menu is shown in Figure 1-5.

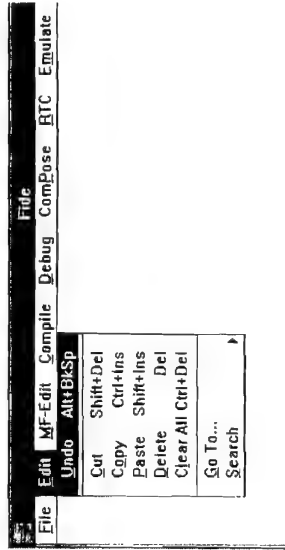


Figure 1-5 Fide/Edit Menu

Undo

The Undo command cancels your previous edit command and restores text to its former state. For example, if you inadvertently delete a selected phrase, you can restore the deleted characters by choosing Undo. One level of Undo is supported. Only the most recent edit command can be undone.

Cut

The Cut command copies a selected block of characters into the Windows clipboard and deletes it from the file image being edited. See the "Full Screen Editing" section below for a description of how to select and move around in text using the mouse and keyboard.

Copy

The Copy command copies a selected block of characters from the edit window to the Windows clipboard. The original text stays intact unchanged in the edit window.

Paste

The Paste command inserts the current contents of the Windows clipboard into the active edit window at the current cursor insertion position. If, at the time the Paste command is chosen, a block of characters is selected in the edit window, it will be overwritten by the Paste text.

Clear All

The Clear All command blanks out all text in the edit window. This command allows you to easily start over. However, the original file is not changed until you choose the Save command.

Go To ...

The Go To ... command brings up a dialog box in which you are asked to enter a line and column. The Go To dialog box is shown in Figure 1-6.

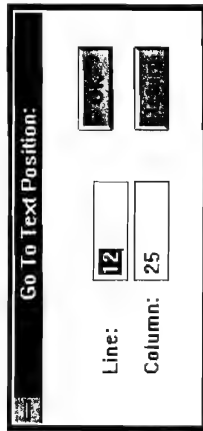


Figure 1-6 Fide/Edit/Go To Dialog Box

Type in the line number and column number to which you wish to position the edit cursor. Fide follows the convention of the first row being referred to as line 0 and the first position of a line referred to as column 0.

Two push buttons are available in the dialog box: (1) OK and (2) return. Pressing the ok button will cause the edit window cursor to move to the specified line and column. The Go To Dialog box, however, remains open ready for a new entry of line and column numbers. Pressing the return button closes the dialog box without affecting cursor position in the edit window.

Search

The Search ... command brings up an hierarchical sub menu containing 3 items:

```
Find ...
Replace ...
Next
```

Find

The Find ... command brings up a dialog box in which you are asked to enter a search string. This dialog box is shown in Figure 1-7.

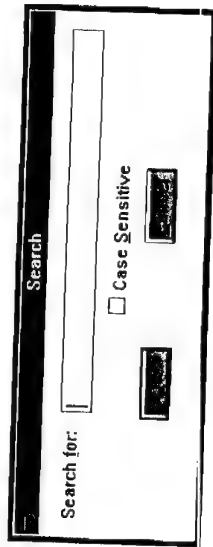


Figure 1-7 Fide/Edit/Search/Find Dialog Box

Replace

The Replace ... command brings up a dialog box in which you are asked to enter both a search string and a replacement string. This dialog box is shown in Figure 1-8.

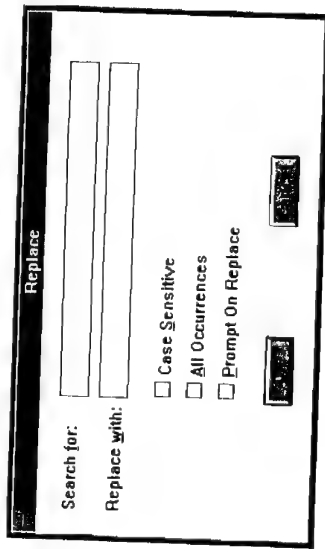


Figure 1-8 Fide/Edit/Search/Replace Dialog Box

Pressing the OK button in either the Find ... or Replace ... dialog boxes positions the cursor to select an occurrence of the search string in the edit

window. In the case of Replace, note that you can choose options to either globally replace or be prompted for replacement at every found instance.

Next

The Next command allows you to proceed with the search, looking for the next occurrence of your search string. You may select the menu option, or you may press the F3 key. For multiple occurrences of a string, it is usually easier to press F3 repeatedly.

Keyboard and Mouse Editing Functions

Fide provides a full complement of text editing functions that can be used through the keyboard and/or mouse. These functions are summarized below:

function	keyboard	mouse
move text cursor	arrow keys	click
beginning of line	home key	click
end of line	end key	click
beginning of document	CTRL-home	
end of document	CTRL-end	
one line up	up arrow	click scroll bar up arrow
one line down	down arrow	click scroll bar dn arrow
one page up	page up key	click on vertical scroll bar
one page down	page down key	click on vertical scroll bar
one word left	CTRL-left arrow	click
one word right	CTRL-right arrow	click
select word		double click
select block	SHIFT-arrow key	click and drag
insert character	SHIFT-click mouse	
delete character	character keys	
	delete or backspace key	

Note that text displayed in Fide edit windows uses a fixed space font (as opposed to a proportional spaced font). This feature allows you to line up text on a column by column basis to assist in visually finding syntax inconsistencies and logic errors.

Window Menu Commands

The Window menu is standard on all Multiple Document Interface (MDI) windows in Microsoft Windows 3.x applications. This menu provides functions to manage the multiple windows that can be brought up in an MDI environment. The Window menu is shown in Figure 1-9.

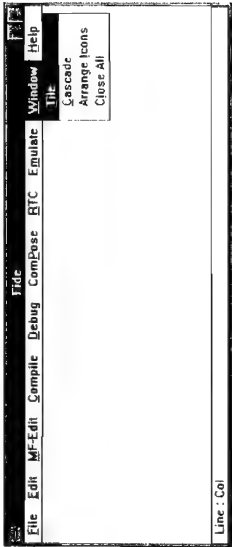


Figure 1-9 Fide/Window Menu

Tile

The Tile command causes all open windows in the main parent window to be resized and positioned so that all are visible and non-overlapping. These windows are sized to fully fill the parent window just above minimized child window icons, if any. An example of tiled windows is shown in Figure 1-10.

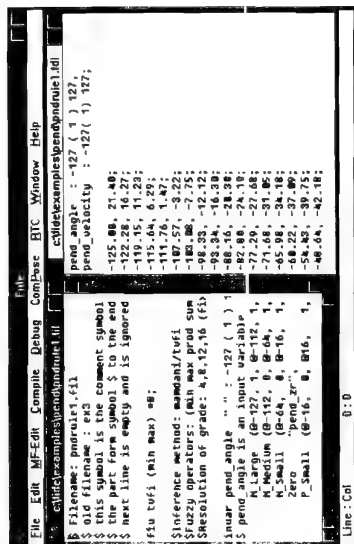


Figure 1-10 Tiled Windows

Cascade

The Cascade command causes all open windows in the main parent window to be positioned and sized in an overlapping sequence of windows. These windows are offset from one another so that caption text of each window is visible. An example of cascaded windows is shown in Figure 1-11.

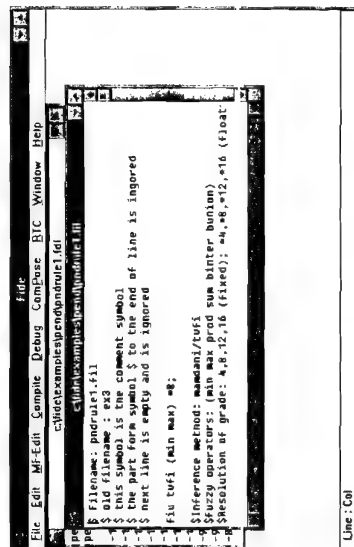


Figure 1-11 Cascaded Windows

Arrange Icons

The Arrange Icon command causes Multiple Document Interface (MDI) icons in open windows which have been minimized to icons to be lined up evenly at the bottom of the MDI window. The icons appear as organized and neat symbols. An example of arranged icons in an MDI window is shown in Figure 1-12.

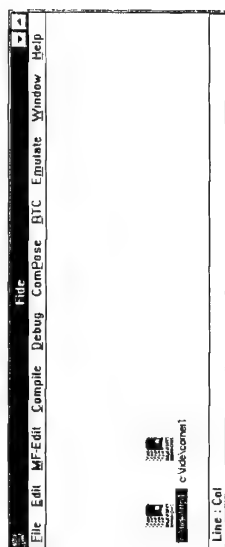


Figure 1-12 Arranged Icons

Clear All

The Clear All command closes all windows, including those in minimized icon form, of a Multiple Document Interface (MDI) window. This command is a short hand way of equivalently selecting each window and explicitly closing the window by double clicking on its control menu. Alternatively, each MDI window can be closed by keying CTRL-F4 successively.

Windows List

This region of the Window menu contains names of open windows in the Multiple Document Interface environment. The currently active window (the topmost window) is indicated by a check mark at the left of its window name. Selecting any window name on the menu makes that window the active one. A typical Window list is shown in Figure 1-13.

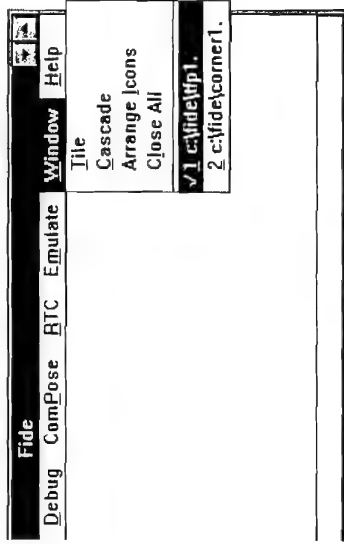


Figure 1-13 Window List Menu

Help

The Help command brings up a Help window. The window contains a list of topics arranged alphabetically. Clicking on any one of these items brings up material on the topic in its own separate window. The Help system is self contained and self instructing so that information on using Help as well as the Fide Help topics themselves are always accessible.

Fide allows you to edit membership functions associated with an open source file. Editing is performed graphically so that you can visually assess the changes you make. Your changes can be stored permanently on disk or just held temporarily in memory to try out "what if" type scenarios.

Perform the following series of Fide commands and actions to display and modify your membership functions.

MF-Edit Command

MF-Edit is a single command pull down menu which launches the Membership Function Editor. Figure 2-1 shows the MF-Edit command.

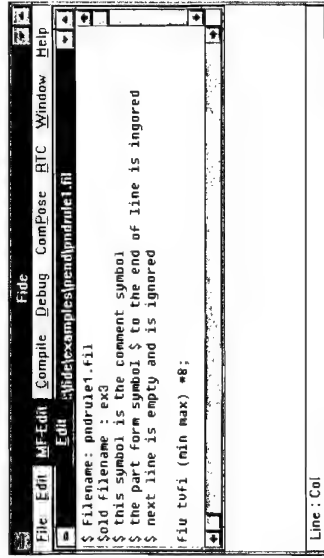


Figure 2-1 Fide/MF-Edit Command

A source file must be open and have been compiled before the Membership Function Editor will operate. MF-Editor edits membership functions which reside in separate files of their own. That is, only those membership functions whose labels in the source refer to a separate file are editable. Those membership functions which are directly defined in the source statements are not editable. For example, consider the following input variable statement:

```
starea ,
N1 180.110, 616.1, 664.01,
N2 180.110, 664.01, 722.01,
N3 180.110, 722.01, 816.110, 8144.01,
P5 180.110, 816.110, 8144.01,
P6 180.110, 8144.01, 816.110, 8144.01,
P7 180.110, 8144.01, 816.110, 8144.01,
Send of Labels
```

The membership function whose label is `pc` is not editable using the MF-Editor; it is fully defined in the source statement itself. To edit it, you must directly change the text or change it to refer to a membership function file. The membership function whose label is `pc` is editable by the MF-Editor. It refers to an external file with the name `pc`. The MF-Editor reads this file for manipulation. The membership function file can be in either point format or lookup table format. See the Fide Inference Language reference for further details on the definition of membership functions.

Select Variable

Choose the Edit command. A dialog box, shown in Figure 2-2, comes up asking you for the variable whose membership functions you wish to edit.

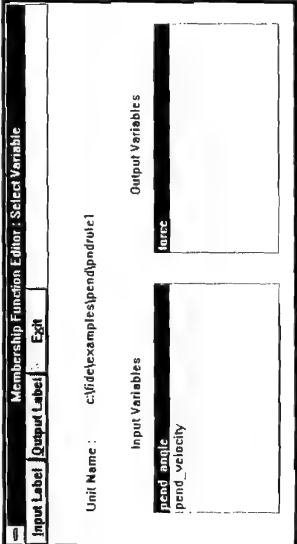


Figure 2-2 Fide/MF-Edit Select Variable Dialog Box

Note that there are two list boxes, the left one for input variables and the right one for output variables. Select a single variable from each list box before initiating the editor for that variable. To select a variable, click on the variable's name. It is highlighted, and all other variables are de-selected. Alternatively you can use the key board to select variables. Press the tab key until you have selected the list box containing variable of interest to you. Use the Up and Down arrow keys to move around the list box, highlighting the variable you wish to select. Once you have selected an input and output variable, press either the Input Label button or the Output Label button.

Input Label

Pressing the Input Label button brings up the Input Membership Function edit display. This display is shown in Figure 2-3.

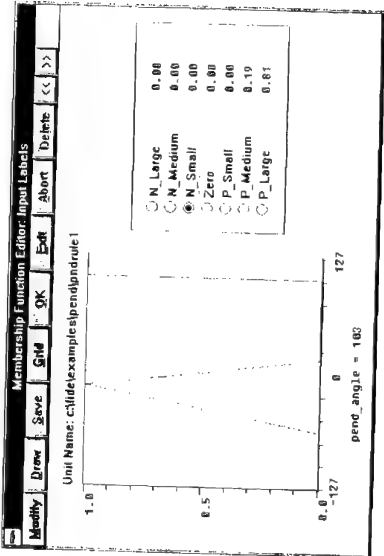


Figure 2-3 Fide/MF-Edit/Input Labels Display

The Input Labels Window serves two purposes: (1) to display the shape and values of membership functions associated with your current Fide Inference Unit, and (2) to allow you to graphically edit the membership functions. This last feature provides considerable power in manipulating the characteristics of your outputs.

As you move the mouse cursor within the bounds of the membership function display, you see your input variable varying in value in a text display just below the main display. In the right half of the window is a boxed display of the membership function values as they vary with input value. Choosing a radio button selects a membership function for editing. The chosen curve is highlighted with the color cyan in the graph display.

Grid Button

Press the Grid button at the top of the display to toggle the grid on and off. Disabling grids could add to clarity to those situations in which membership functions are densely packed together.

Modify Button

There are several ways to modify your membership functions. The first step is to press the modify button at the upper left corner of the window. If you had chosen a label whose membership function is not editable in the MF-Editor, the dialog box shown in Figure 2-4 appears.



Figure 2-4 Not Editable MF Dialog Box

Fide asks if you want to proceed even if not editable. If you chose OK, the MF-Editor allows you to make modifications to your membership function.

However, changes so made are stored in memory only. They cannot be stored permanently on disk. In other words, your modifications are valid only for the current session of Fide. Once you bring up another Fide Inference Unit (source file), memory is released and your membership function changes are no longer in effect. However, this approach is effective for trial situations where you would like to explore variations in design as quickly and efficiently as possible.

Label Dialog Box

Whether you have selected an editable or non-editable membership function, the next dialog box encountered is shown in Figure 2-5 below.

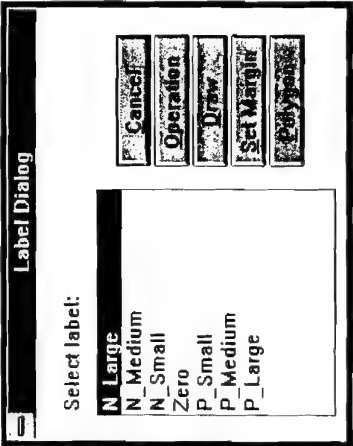


Figure 2-5 Fide/MF-Edit/Label Dialog Box

The label list box on the left is intended primarily for the *Operation* (see Operation section below) function of MF-Editor. However, you may also use it to change your selection of membership function for editing. The editability

constraints described above still apply even if a membership function is selected through this means. The push buttons are described below.

Cancel Button

Cancel allows you to return to the prior window (MF-Edit Input Labels Display) without any action taken on a selection you may have made in this dialog box.

Operation Button

MF-Edit supports a powerful tool for membership function editing. You can perform standard operations on existing membership functions to produce a resultant membership function. Press the Operation button of the Label Dialog box as shown in Figure 2-5 above to initiate an operation. An Operator Dialog box comes up as shown in Figure 2-6.

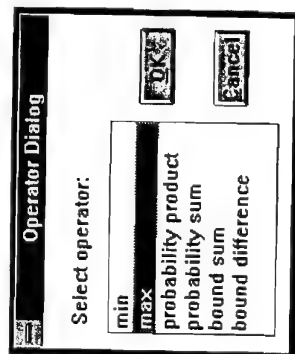


Figure 2-6 MF-Edit/Operator Dialog Box

There are 6 operations supported. They are:

min
max
 $x \wedge y$
 $x \vee y$

probability product
probability sum
bound sum
bound difference
 $(x+y) \wedge 1$
 $(x+y-1) \vee 0$

where: x = operand 1
 y = operand 2
 \wedge = minimum
 \vee = maximum

To use an operation, perform the following steps:

1. Select the operation by clicking on one of the listed operations. The operation is highlighted. Press the OK button to choose that operation.
2. An Operand 1 dialog box similar to that shown in Figure 2-5 above appears. Click on a label name to select it as operand 1. Press the OK button to choose the membership function represented by the label name.
3. An Operand 2 dialog box appears. Click on a label name to select it as operand 2. Press the OK button to choose the membership function represented by the label name.
4. You are brought back into the Input Labels dialog box, where the resultant membership function and the original membership function is highlighted in cyan. You must now confirm this operation by pressing the OK button or cancel the operation by pressing the Abort button. Abort brings you back to the prior state. Press OK to confirm your operation and replace the former membership function with the updated one.

Draw Button

The Label Dialog (Figure 2-5) allows you to select a particular membership function for editing. Use your mouse to click on a label. It is highlighted in the

list box. Press the draw button. MF-Edit places itself in edit mode and disables the vertical hairline cursor. You are now in a position to edit the curve.

MF-Edit provides two methods of editing. The first is simply to click your mouse at points where you want to draw lines from adjacent points. This is shown in Figure 2-7. You can continue clicking until you have formed the shape you want.

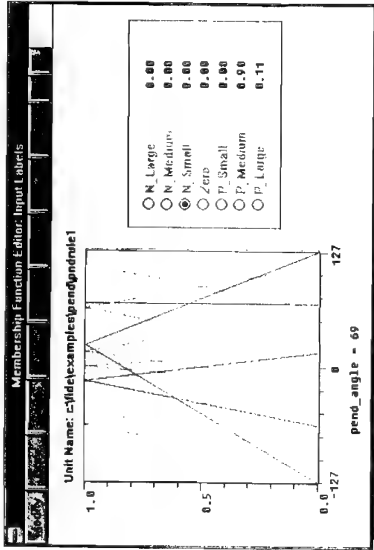


Figure 2-7 Fide/MF-Edit/Draw/Point & Click

Note that when clicking, lines are drawn to the click (new) point from the closest established points on either side of the new point. This technique parallels the text point representation of membership functions, where the @point syntax corresponds to the point at which you click. Note, however, that using point and click here does not modify your @point source.

The second of method of editing a membership function is free form drawing. To initiate this, click and drag your mouse in the direction you wish to shape your new membership function. The curve will follow your mouse movement as you slowly drag it across the membership function space.

Once you are satisfied with your modifications, you can either press the OK button or the Abort button. The OK button implements in memory all changes you have made. Abort cancels all your previous effort and returns you to the prior state. Figure 2-8 shows an example of a free form membership function drawing session.

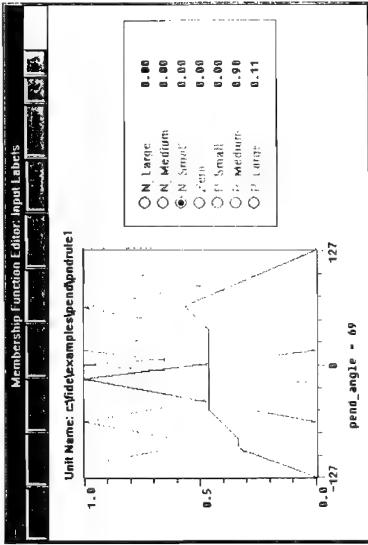


Figure 2-8 Fide/MF-Edit/Draw/Free Form

If you pressed OK, your new membership function is in effect. However, it is still not permanently modified in its disk file. To do so press the Save button at the top of the Window. MF-Edit stores your membership function in lookup table format. Now you can use for future sessions of Fide development. If you had selected a non-editable membership function, the Save button would be disabled (grayed out), and you would not be able to store your changes to disk. This limitation is the only difference between an editable and non-editable membership function.

Set Margin Button

Press the Set Margin button to place your membership function display into set margin mode. In this mode, you may click and drag either or both the left and right boundaries of the membership function graph. This serves to restrict the area of the graph on which you may make editing changes. Membership functions or portions of membership functions outside these bounds remain static while you edit within the bounds. Once you have reconfigured margins, press the Draw button to redraw your selected membership function. It now conforms at the edges to the boundaries you have set. Figure 2-9 illustrates editing within a narrower boundary region.

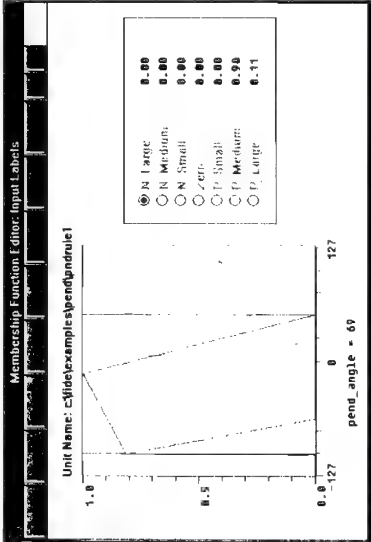


Figure 2-9 Fide/MF-Edit/Set Margin

Polygon Button

Press the Polygon button to activate MF-Edit's polygon membership function editing capabilities. The Polygon editing function allows you to specify precisely the vertices of polygonal membership function. An enhanced display

with text boxes for polygon coordinate entries comes up as shown in Figure 2-10.

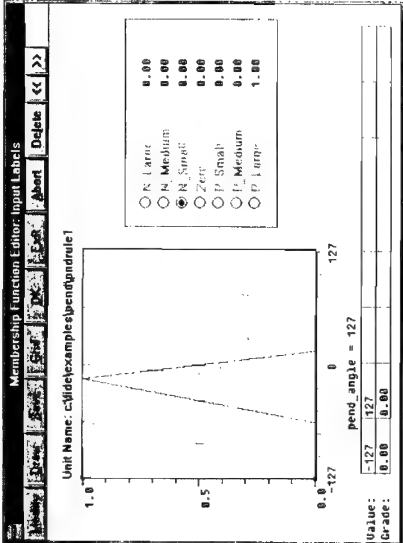


Figure 2-10 Fide/MF-Edit/Polygon Edit Display

There are two means of editing the polygon display: (1) graphically and (2) textually. Graphics editing is quick and intuitive. Textual editing is precise and informative. Modifications made in one mode is automatically reflected in the other mode. There is no loss of information.

Graphics Editing

To edit graphically, click your mouse in the graphical display region. Each click enters a point into the membership function. The entered point is considered the active point, denoted by a small blue square surrounding the point. This is shown in Figure 2-11.

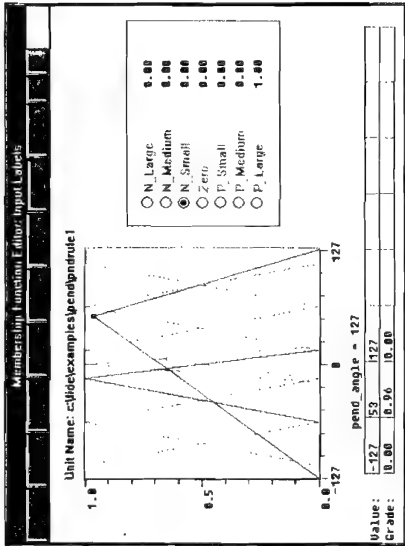


Figure 2-11 Fide/MF-Edit/Polygon Graphical Edit

Note that the text boxes at the bottom of the display have also been updated to include a new point entry. The points are inserted in graphical order. Up to 20 points can be so created and inserted.

Once points are entered into the display, you can move the points graphically. Select a point by clicking on a vertex of the polygon and dragging it to a new position. Note that you are limited to the region formed by the vertical boundaries of the two points adjacent to your selected point. You can also delete a point by making it active (click on it) and pressing the Delete push button.

Text Entry Editing

To edit textually, click on one of the text boxes at the bottom of the display. A dialog box comes up allowing you to edit a point textually as shown in Figure 2-12.

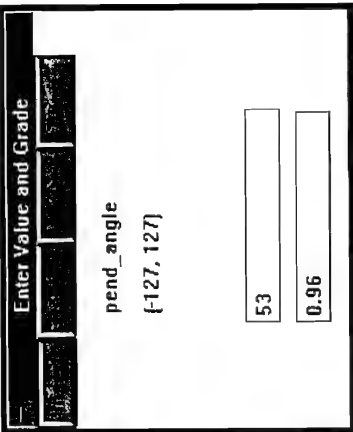


Figure 2-12 Fide/MF-Edit/Polygon Textual Edit

Double click on a text edit box located at the bottom of the dialog box. This selects the text for editing. Alternatively, you can press the tab key until you have selected the text edit box of your choice. Select the text by pressing the SHIFT-arrow combination. Once selected, type in the numbers you wish to enter. Note that the variable range is displayed above the edit boxes. Entered values should be within this range. Grade values are always between 0 and 1.

The push buttons at the top of the dialog box have the following meanings:

- Insert insert a new point at the position specified by your text entries
- Delete remove the current active point
- Move reassign the active point to the position specified by your text entries.
- Abort go back to the MF-Edit display without implementing any modifications you may have made in this dialog box.

An example of an inserted point is shown in Figure 2-13.

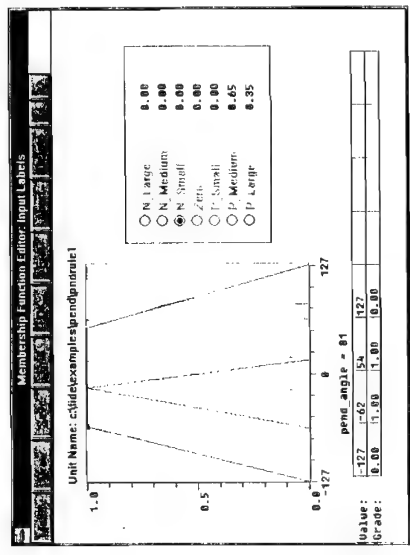


Figure 2-13 Fide/MF-Edit/Polygon/Insert Point

Function Push Buttons

The << and >> push buttons at the top of the window allows you to scroll the text boxes to the left and right respectively. Twenty points can be entered and displayed graphically. However, only eight text boxes are available for numeric display.

Once you have entered your membership function points, press the OK button to confirm the polygon to Fide, or press Abort to cancel the editing session. The Delete button allows you to delete the active point, as explained above.

Output Label

Press the Output Label button (Figure 2-2) to bring up a graphic display of the output labels associated with your fuzzy inference unit. There are two types of Output Label displays: (1) TVFI - not editable, and (2) Mamdani - editable.

The TVFI (Truth Value Fuzzy Inference) inference method provides for singleton membership functions and can only be edited in the source file using a text editor. The Mamdani inference method supports membership functions similar to those used for input variables, and can be edited using all the tools available for input label editing. See the Fide Language Reference manual for information on how to specify either the TVFI or Mamdani methods.

TVFI Inference Method

If the TVFI method is specified, the Output Labels window displays read only information. The Modify button in the upper left corner is disabled (grayed out). A sample display of the MF-Edit Output Labels window for TVFI processing is shown in Figure 2-14.

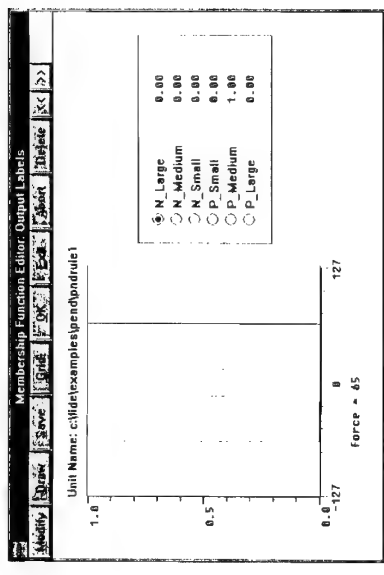


Figure 2-14 Fide/MF-Edit/Output Labels Window - TVFI

You cannot modify the membership functions as you can with input labels. As you move your mouse cursor in the graph display region, note that for varying output values displayed at the bottom of the graph, corresponding membership function values are displayed in the text box to the right of the graph. In the

Chapter 3 Compiling Source

example shown above in Figure 2-14, the membership functions are singletons (point values).

Mamdani Inference Method

If the Mamdani method is specified, the Output Labels window acts identically to the Input Labels window. The Modify push button is enabled, and you can edit the membership functions of an output label either graphically or textually. See the description of Input Label processing above for more information. A sample display of the MF-Edit Output Labels window for Mamdani processing is shown in Figure 2-15.

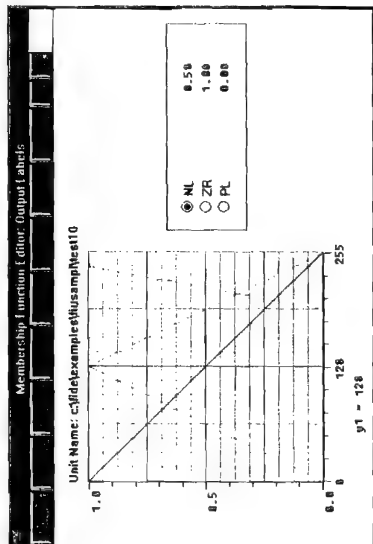


Figure 2-15 Fide/MF-Edit/Output Labels Window - Mamdani

The display grid can be toggled on or off by pressing the Grid button at the top row of buttons. Other buttons there are grayed out indicating those functions are disabled.

Fide compiles source statements into object code. The object code, comprising multiple files, is usable to Fide for other functions within Fide. The Debugger, Composer, and Real Time Code Generator all require data generated from the Fide Compiler. Hence compiling source is a prerequisite for other activities in the development process.

The compile process is straightforward. You choose the Compile command. The bulk of your work, however, is in writing source statements for compilation. Fide's Language Reference manual provides instructions, syntax, and examples for crafting your fuzzy inference statements. The Fide compiler assists you in source statement development by identifying syntax errors during the compilation. You can automatically trace back to the source statement, and re-edit the offending section of code.

Compile Menu Commands

The Compile menu initiates Fide compilers to perform translation of source statements into object code. This pull down menu is shown in Figure 3-1.

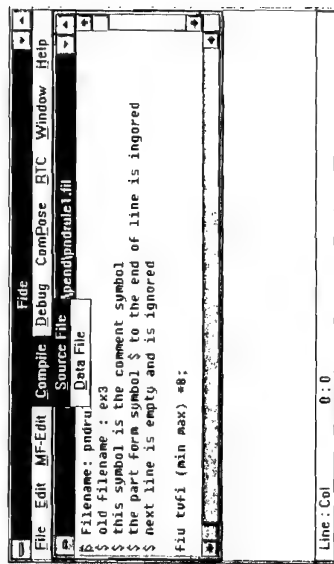


Figure 3-1 Fide/Compile Menu

Source File

The Source File command invokes Fide to compile the file shown in the active edit window. Note that a source file should have .FIL as its DOS filename extension. While Fide is compiling, a message box is brought up with the text "Compiling ..." displayed.

Upon successful completion of compilation, a further message box is displayed indicating the compilation was successful. This message is shown in Figure 3-2.



Figure 3-2 Fide/Compile/Success Message Box

Once Fide Compile processing has completed, a Messages window appears immediately below the edit window whose file was just compiled. If the compilation completed without error, there will be no error messages appearing in the Messages window. However, if any problems (e.g. syntax errors, special conditions) were encountered, a message for each problem will be displayed in the Messages window. Each message occupies one line of the Messages window. A typical Messages window with several messages appearing is shown in Figure 3-3.

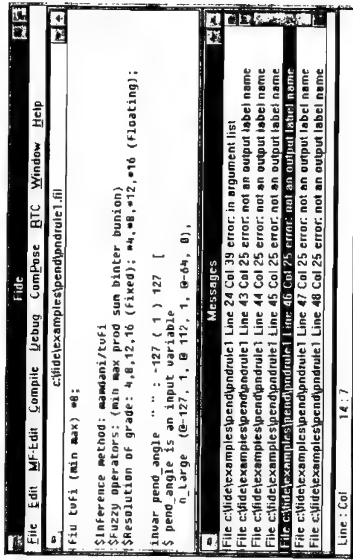


Figure 3-3 Fide/Compile/Messages Window

You can easily track down the errors specified by these messages by double clicking on an error message. If the message has a line and column number embedded in its text (the Fide Compiler is designed to include these numbers in generated error messages), Fide automatically positions the edit window text cursor to select the line and column of your message. You are placed in your edit window ready for corrections to be made. You can go back to the Messages Window at any time (click anywhere on the window) to select the next error message.

Data File

The Data File command invokes the Fide Simulator Input Source Compiler. Note that a data file should have .FDL as its DOS filename extension. The Data file must be open in an edit window. (This is true the other way round as well, i.e. you must open the source rule file before source compilation can take place.) If you attempt to compile data without a data file open, a warning message appears as shown in Figure 3-4.



Figure 3-4 *Incorrect File For Compile*

Input data is used to drive input variables in the Simulation module of Debugger (see chapter 4 Debugging and Analyzing). When compiling begins, a message box with the statement "Input Compiling..." is displayed.

On successful completion of the compilation, a further message box indicating success is displayed. This is shown in Figure 3-5.



Figure 3-5 *Fide/Compile/Input/Success Message Box*

After the input compilation has completed, a Messages Window (identical to that of the Source Compilation) comes up displaying any problems encountered in the compilation. To track down errors in the Input source file, double click on a message line in the Messages window. Fide will position the cursor to the source file line referred to by the error message. There must be a line and column number embedded in the error message for this automatic tracing to take place. However, Fide's Input Compiler is designed to provide the

line and column number with nearly all compiler messages. (see Figure 3-3 under the section "Source File" for an example of a Messages window display.)

Chapter 4 Debugging And Analyzing

The Debug command opens up a rich set of tools to track down design and logic errors not caught at compile time. Moreover, the tools provide an intuitive means of visualizing or modeling your fuzzy inference unit that maps input to output over the range of your application. The following sections describe the Fide commands by which you access and specify these tools

Debug Command

The Debug menu command initiates a separate window to manage the tools available for analyzing and debugging your Fide Inference Unit. The Debug command is shown in Figure 4-1. Note that a source file must be open before the Debugger can begin operation. If there are multiple source files open, the active file, i.e. the window with its caption bar highlighted, is the one acted upon by Debugger.

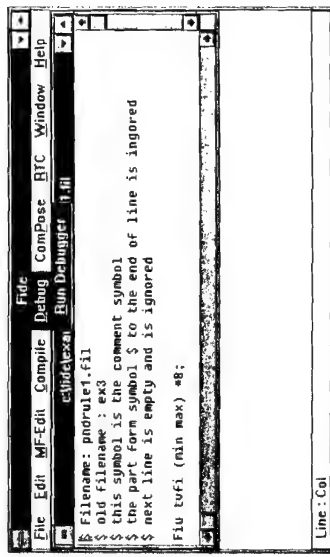


Figure 4-1 Fide/Debug Command

Fide Debugger

The Debugger is a set of Fide functions that encompasses more than debugging in the traditional sense of the term. You use it not only to find and correct errors in your fuzzy inference design logic, but you can also use it to analyze and fine tune your final design. It provides tools for validity checking, optimization, stability analysis, and reliability projection.

The Debug command launches a separate Multiple Document Interface (MDI) window to manage the tools available for analyzing and debugging your Fide Inference Unit. This window is shown in Figure 4-2.

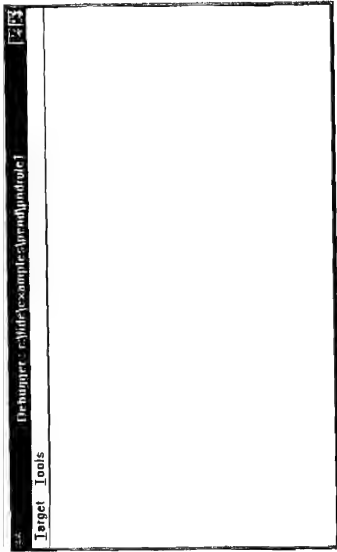


Figure 4-2 Debugger Window

Each of the menus of Debugger are discussed below.

Target

The Target menu, shown in Figure 4-3 below, is identical in format to the RTC menu of the Fide Window (see Figure 5-1). The Motorola chips supported by Fide are listed in a hierarchical submenu. Selecting one of these menu items

sets the Fide Debugger environment to the specific characteristics of the chip chosen. For example, simulation display of input and output curves take on timing characteristics unique to a particular chip.

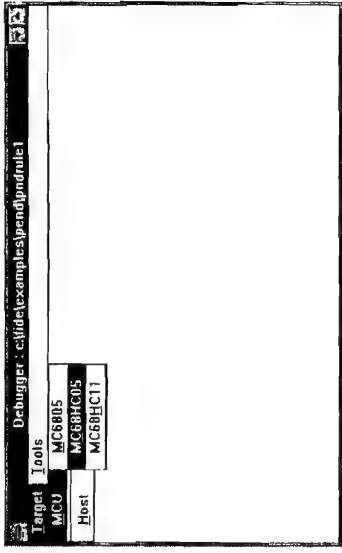


Figure 4-3 Debugger/Target Menu

Choosing the Host command sets the Debugger to an IBM PC compatible environment. In other words, Fide assumes you will run your fuzzy inference application on a PC platform very similar to the one on which you are developing your application. If a target is not selected before debugging begins, Fide assumes the Host selection as default.

Tools

The Tools menu lists the three major debugging tools available in Fide. They are the (1) Tracer, (2) Analyzer, and (3) Simulator. Each is explained fully below.

Tracer

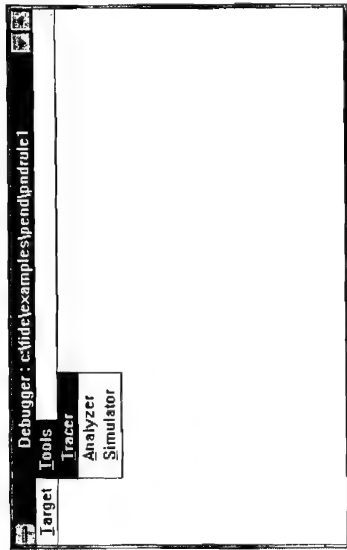


Figure 4-4 Tracer Command

The Tracer function, whose initiating command is shown in Figure 4-4, allows you to isolate a user-defined portion of your Fide Inference Unit input/output transfer function and quickly trace back to the rule statements in source responsible for transfer behavior. Along the way, Tracer provides graphical tools to enhance your understanding of the fuzzification and defuzzification process applied to your inference unit. Tracer also allows you to edit the membership functions using freehand graphical drawing techniques.

Select Variable

Choosing the Tracer command in the Debugger window, The Select Variable Dialog Box comes up. This dialog box is shown in Figure 4-5.

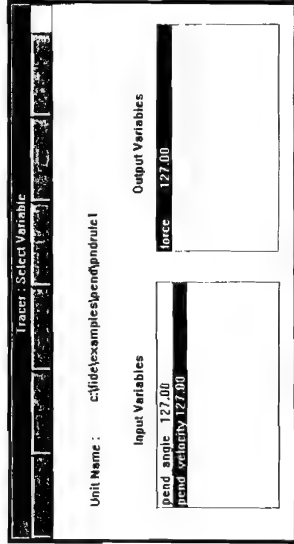


Figure 4-5 Tracer/Select Variable Dialog Box

This dialog box allows you to select the input and output variables you wish to trace. Select items by clicking on the input variable and output variable of your choice. Alternatively you may use your keyboard to select the variables. Use the tab key to cursor over to a list box, and then use the up or down arrow keys to select a variable.

Input variables and their values are displayed in the left list box, while output variables and their values are displayed in the right list box. When selected, the entire line of the variable is highlighted in the list box. Note that only one variable each for input and output can be selected at any one time.

The dialog box contains a row of push buttons at the top of the window. These are used to further refine your selection before tracing.

Set Input Value

Press the Set Value button to bring up the Set Input Value dialog box as shown in Figure 4-6.

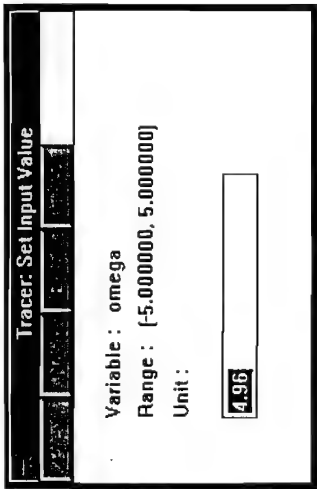


Figure 4-6 Tracer/Set Input Value Dialog Box

This dialog box allows you to enter an initial value for all input variables. Press the Next button or the Previous button to store the entered value into the variable and bring up the next or previous variable. The Cancel button aborts the current value entry. It does not abort values previously entered. Press the Enter button to store the entered value and return to the Select Variables dialog box.

All input variables must be initialized to valid values before tracing can begin. The range of valid values is displayed in the dialog box. Tracer places default initial values in the variables when the Tracer is first invoked. You should, however, override these entries with your own values, defining points of interest to you.

Input Labels

Press the Input Label button to bring up the Tracer/Input Label Window. This window is shown in Figure 4-7.

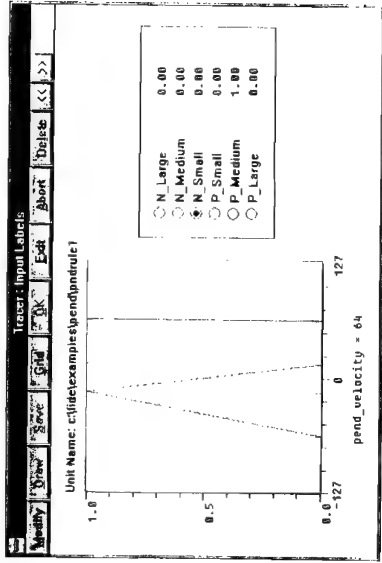


Figure 4-7 Tracer/Input Labels Window

The Input Labels Window serves two purposes: (1) to display the shape and values of membership functions associated with your current Fide Inference Unit, and (2) to allow you to graphically edit the membership functions before tracing begins. This last feature provides considerable power in manipulating the characteristics of your outputs.

The functions here are identical to the ones available in the MF-Edit function of the main Fide window (see chapter 2) with one major exception. The Tracer membership function editor does not allow permanent storage of modified membership functions to disk. The intent is to provide membership function prototyping to investigate and try out different possibilities in the context of Fide debugging and analysis tools.

Label Buttons

As you move the mouse cursor within the bounds of the membership function display, you see your input variable varying in value in a text display just below the main display. In the right half of the window is a boxed display of the

membership function values as they vary with input value. The label whose radio button is selected represents the selected membership function (highlighted in the color magenta in the graph) upon which editing can be applied.

Grid

Press the Grid button to toggle the grid on and off. Disabling grids could add to clarity for those situations in which membership functions are densely packed together.

Modify

There are several ways to modify your membership functions. The first step is to press the modify button at the upper left corner of the window. Doing so brings up the Label Dialog box as shown Figure 4-8.

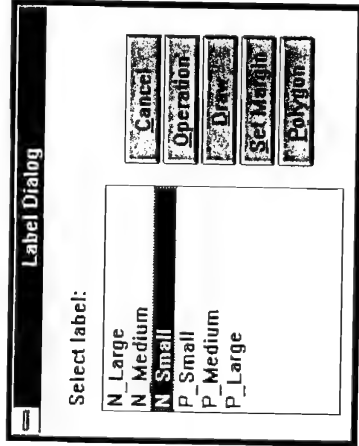


Figure 4-8 Tracer/Label Dialog Box

Draw Button

The Label Dialog allows you to select a particular membership function for editing. Use your mouse to click on a label. It is highlighted in the list box. Press the draw button. Tracer places itself in edit mode and outlines in red the membership curve you selected and grays out all remaining curves. You are now in a position to edit the curve.

Tracer provides two methods of editing. The first is simply to click your mouse at the point where you want to draw lines from adjacent points. This is shown in Figure 4-9. You may continue clicking until you have formed the shape you want.

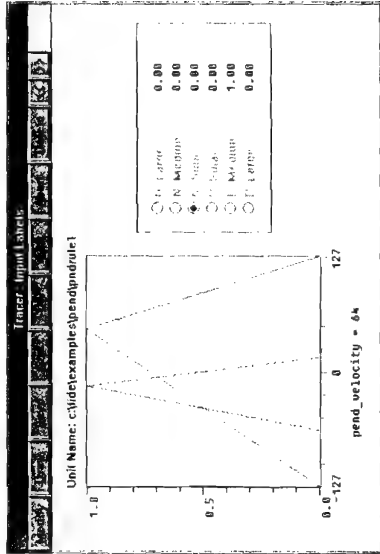


Figure 4-9 Tracer/Membership Function Editing: Point & Click

The second of method of editing a membership function is free form drawing. To initiate this, click and drag your mouse in the direction you wish to shape your new membership function. The curve will follow your mouse movement as you slowly drag it across the membership function space.

Once you are satisfied with your modifications, you can either press the OK button or the Abort button. The OK button implements in memory all changes you have made. Abort cancels all your previous effort and returns you to the prior state. Figure 4-10 shows an example of a free form membership function drawing session.

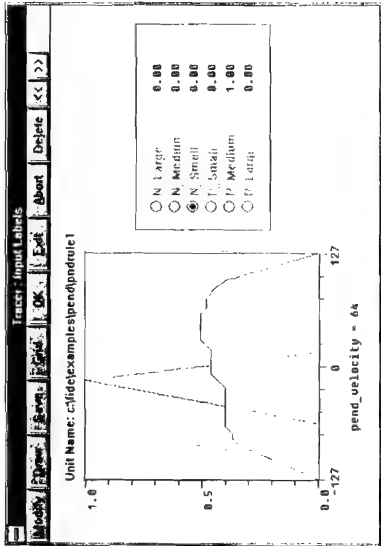


Figure 4-10 Tracer/Membership Function Editing: Free Form Draw Operation Button

Tracer supports one further tool for membership function editing. You may perform standard operations on existing membership functions to produce a resultant membership function. Press the Operation button of the Label Dialog box as shown in Figure 4-8 above to initiate an operation. An Operator Dialog box comes up as shown in Figure 4-11.

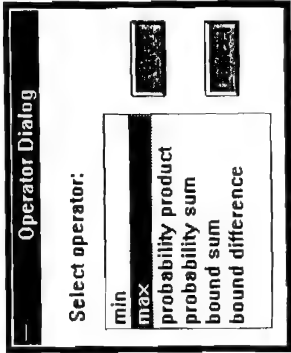


Figure 4-11 Tracer/Operator Dialog Box

There are 6 operations supported. They are:

- | | |
|---------------------|----------------------|
| min | $x \wedge y$ |
| max | $x \vee y$ |
| probability product | xy |
| probability sum | $x + y - xy$ |
| bound sum | $(x + y) \wedge 1$ |
| bound difference | $(x + y - 1) \vee 0$ |

where: x = operand 1
 y = operand 2
 \wedge = minimum
 \vee = maximum

To use an operation, perform the following steps:

1. Select the operation by clicking on one of the listed operations. The operation is highlighted. Press the OK button to choose that operation.

2. An Operand 1 dialog box similar to that shown in Figure 4-11 above appears. Click on a label name to select it as operand 1. Press the OK button to choose the membership function highlighted by the label name.
3. An Operand 2 dialog box appears. Click on a label name to select it as operand 2. Press the OK button to choose the membership function highlighted by the label name.
4. You are brought back into the Tracer/Input Labels dialog box, where the resultant membership function is highlighted in red. You must now confirm this operation by pressing the OK button or canceling the operation by pressing the Abort button. Press Abort to bring you back to the prior state. Press OK to confirm your operation and change the color of the resultant membership function to the color of the original selected membership function.

Set Margin Button

Press the Set Margin button to place your membership function display into set margin mode. In this mode, you may click and drag either or both the left and right boundaries of the graph. This serves to restrict the area of the graph to which you may make editing changes. Membership functions or portions of membership functions outside these bounds remain static while you edit within the bounds.

Polygon Button

Press the Polygon button to activate Fide's polygon membership function editing capabilities. The Polygon editing function allows you to specify precisely the vertices of polygonal membership function. An enhanced display with text boxes for polygon coordinate entries comes up as shown in Figure 4-12.

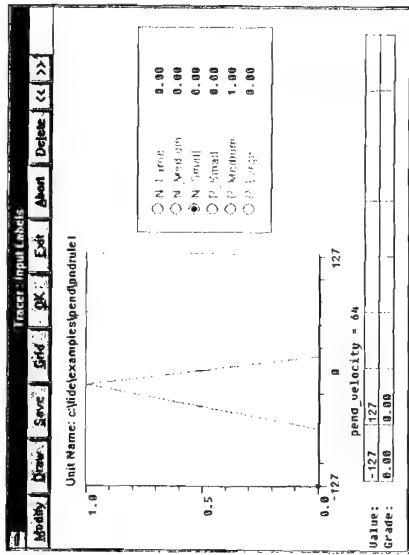


Figure 4-12 Tracer/Label/Polygon Edit Display

There are two means of editing the polygon display: (1) graphically and (2) textually. Graphics editing is quick and intuitive. Textual editing is precise and informative. Modifications made in one mode is automatically reflected in the other mode. There is no loss of information.

Graphics Editing

To edit graphically, click your mouse in the graphical display region. Each click enters a point into the membership function. The entered point is considered the active point, denoted by a small blue square surrounding the point. This is shown in Figure 4-13.

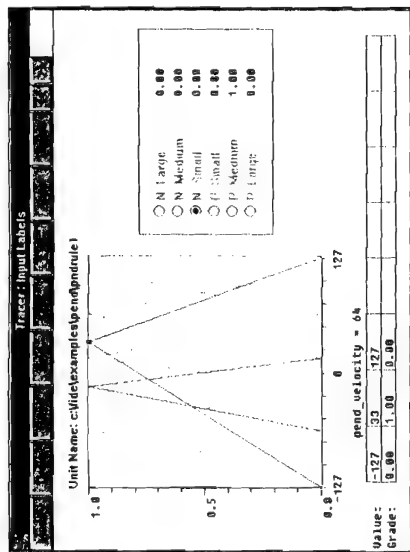


Figure 4-13 Tracer/Label/Polygon Graphical Edit

Note that the text boxes at the bottom of the display have also been updated to include a new point entry. The points are inserted in graphical order. Up to 20 points can be so created and inserted.

Once points are entered into the display, you can move the points graphically. Select a point by clicking on a vertex of the polygon and dragging it to a new position. Note that you are limited to the region formed by the vertical boundaries of the two points adjacent to your selected point. You can also delete a point by making it active (click on it) and pressing the Delete push button.

Text Entry Editing

To edit textually, click on one of the text boxes at the bottom of the display. A dialog box comes up allowing you to edit a point textually as shown in Figure 4-14.

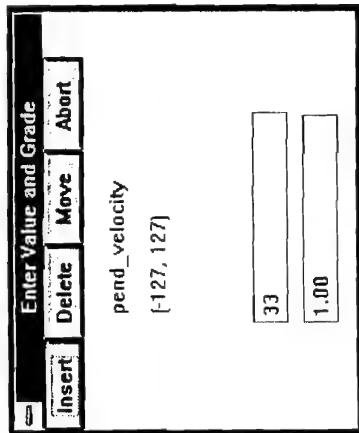


Figure 4-14 Tracer/Label/Polygon Textual Edit

Double click on a text edit box located at the bottom of the dialog box. This selects the text for editing. Alternatively, you can press the tab key until you have selected the text edit box of your choice. Select the text by pressing the SHIFT-arrow combination. Once selected, type in the numbers you wish to enter. Note that the variable range is displayed above the edit boxes. Entered values should be within this range. Grade values are always between 0 and 1.

The push buttons at the top of the dialog box have the following meanings:

- Insert insert a new point at the position specified by your text entries
- Delete remove the current active point
- Move reassign the active point to the position specified by your text entries.
- Abort go back to the MF-Edit display without implementing any modifications you may have made in this dialog box.

An example of an inserted point is shown in Figure 4-15.

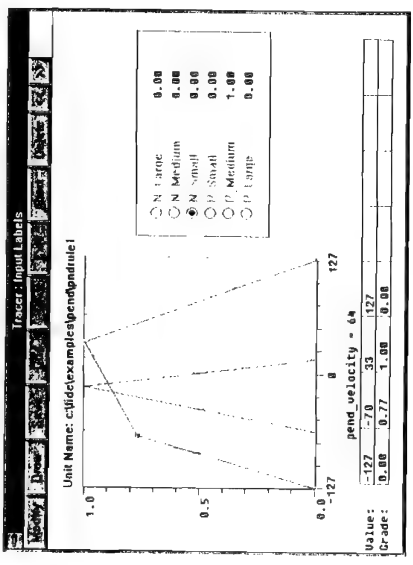


Figure 4-15 Tracer/Label/Polygon/Insert Point

Function Push Buttons

Once you have entered your membership function points, press the OK button to confirm the polygon to Fide, or press Abort to cancel the editing session. The Delete button allows you to delete the active point, as explained above.

Once you have entered your membership function points, press the OK button to confirm the polygon to Fide, or press Abort to cancel the editing session. The Delete button allows you to delete the active point, as explained above.

Execute

Press the Execute button to initiate computation of the output values and generation of intermediate data needed to perform a trace of your fuzzy inference unit in its current state. Values entered for input variables and

modifications made to membership functions are used to compute the output values you see in the output variables list box, an example of which is shown in Figure 4-5.

Output Label

Press the Output Label button to bring up a graphic display of the output labels associated with your fuzzy inference unit. The Tracer Output Labels window is shown in Figure 4-16.

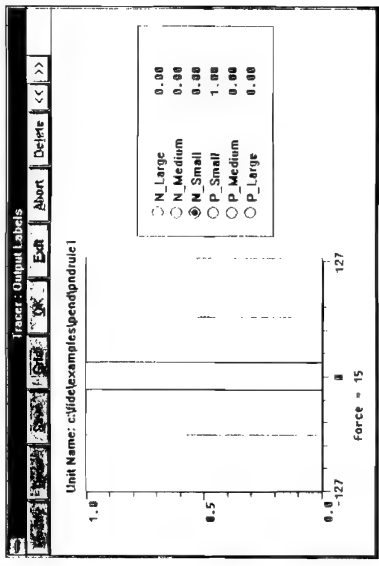


Figure 4-16 Tracer/Output Labels Window

There are two types of Output Label displays: (1) TVFI - not editable, and (2) Mamdani - editable. The TVFI (Truth Value Fuzzy Inference) inference method provides for singleton membership functions and can only be edited in the source file using a text editor. The Mamdani inference method supports membership functions similar to those used for input variables, and can be edited using all the tools available for input label editing. See the Fide

Language Reference manual for information on how to specify either the TVFI or Mamdani methods.

The Output Labels window shown in Figure 4-16 is an example of the TVFI method. It displays read only information. You cannot modify the membership functions as you can with input labels. As you move your mouse cursor in the graph display region, note that for varying output values displayed at the bottom of the graph, corresponding membership function values are displayed in the text box to the right of the graph. In the example shown above in Figure 4-16, the membership functions are singletons (point values).

The display grid can be toggled on or off by pressing the Grid button at the top row of buttons. Other buttons there are grayed out indicating those functions are disabled for output labels.

Trace

Press the Trace button to launch a series of dialog boxes that narrow down the scope of trace to particular elements of your fuzzy inference unit.

Output Membership Function Window

The first dialog box to come up is the Tracer/Output Membership Function window. This window is shown in Figure 4-17.

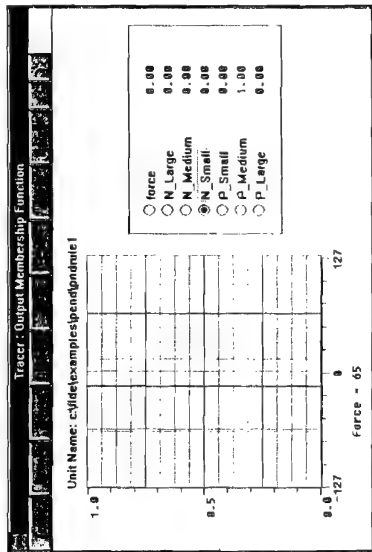


Figure 4-17 Tracer/Output Membership Function Window

This window, like the Output Label window in Figure 4-16 above, displays output membership functions of the current fuzzy inference unit. Unlike the Output Label display, this window highlights only those membership functions which contribute towards the selected output value.

The display is for information purposes only. Output membership functions cannot be modified. As you move your mouse in the membership function region, note that as the output value varies at the bottom of the graph, corresponding membership function values in the text box to the right changes as well. Press the Grid button in the top row of buttons to toggle the display grid on and off.

Select Output Label

Press the Trace button in the Output Membership Function window to bring up the Tracer/Select Output Label dialog box. This dialog box is shown in Figure 4-18.

Select Rules

Press the Trace button in the Select Output Label window to bring up the Tracer/Select Rules Dialog box. This window is shown in Figure 4-19.

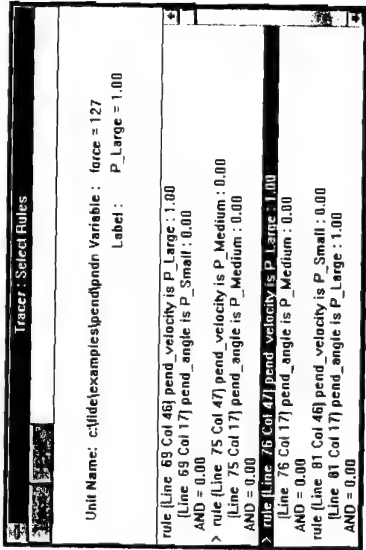


Figure 4-19 Tracer/Select Rules Window

Summary information is shown at the top of the window. The rules contributing to the membership function grade are shown in a scrollable list at the bottom of the window. Note that each rule has associated with it a value. This value is used in minimization (AND) operations and maximization (OR) operations to arrive at the membership grade. Rule pairs are minimized, whose resultant value is maximized against all other pair minimizations to arrive at the desired label value. This process is followed in ordered sequence by rules listed in the scroll box.

To trace a rule to its source, click on the rule. More than 1 rule can be so selected. A check mark is placed on the left side of all rules selected. After you have selected all the rules you want, press the Trace button at the top of the window to launch the trace.

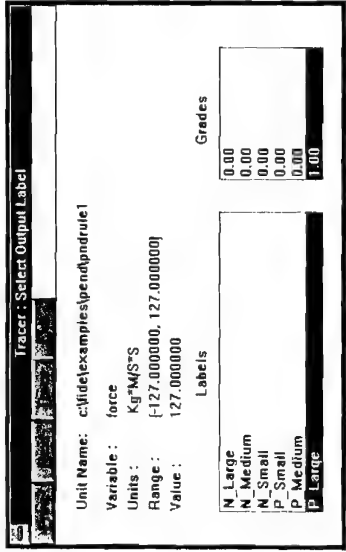


Figure 4-18 Tracer/Select Output Label Window

This window can be thought of as a text version of the previously described Output Membership Function window. The left list box displays label names of output membership functions. The right list box displays corresponding grades (contribution values) for the current state of your inference unit.

Select the membership function value you are going to trace by clicking on the label name or grade value. The label name and grade value are linked so that highlighting one (selecting) also highlights the other. You may select only one output label at a time.

The Next and Prev buttons at the top row of the window allows you to select alternative output variables without having to back up to the Select Variables dialog box. If a new output variable is selected, membership function labels and grades change to reflect the new variable.

Jump To Source

Pressing Trace causes all Debugger windows to close and a Messages window to open. The Fide window environment becomes active. The Messages window opens directly beneath the source file edit window displaying the rules of your fuzzy inference unit. In the Messages window itself, note that the rule statements you selected in Tracer are now listed. To trace to the source rule brings you into the appropriate edit window and positions the cursor at the line and column indicated by the message statement. This last step in tracing is identical to that of identifying errors in the compile step of the Fide development process (see chapter 3). Clicking on any line in the Messages window, which has embedded in it a line and column number produces the same action: transfer to the specified position in the edit window. An example of a Messages window containing rule statements coming from the Tracer is shown in Figure 4-20.

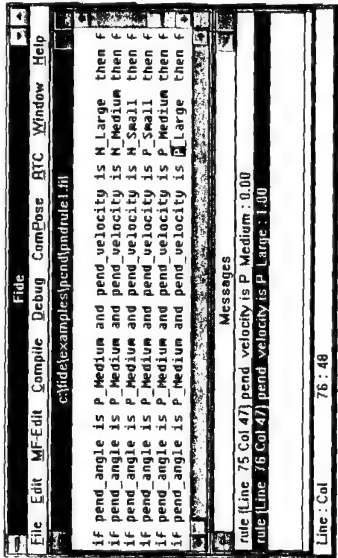


Figure 4-20 Fide/Messages Window with Rules From Tracer

Analyzer

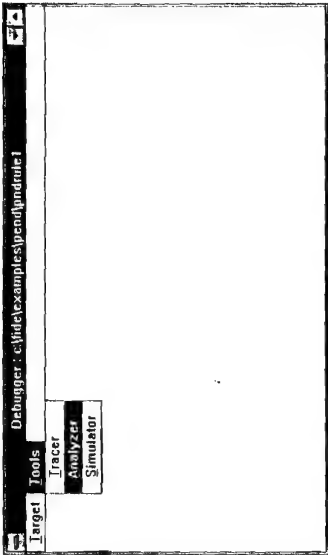


Figure 4-21 Analyzer Command

The Fide Analyzer, whose initiating command is shown in Figure 4-21, provides visualizations of your fuzzy inference unit. Analyzer presents several graphic representations of the fuzzy unit transfer function. The transfer function can be thought of as the range of outputs resulting from every combination of inputs over the domain of inputs. It is time invariant and represents the characteristic function of a given fuzzy inference unit. In addition to visualizing a Fide Inference unit in graphic format, Analyzer can also automatically trace a point from its graphic representation to its source text statement. This feature facilitates trying out "what if" scenarios, as well as debugging errant source statements.

Select Variables

Choose the Analyzer command in the Debugger window. The Analyze/Select Variables Dialog Box comes up. This dialog box is shown in Figure 4-22.

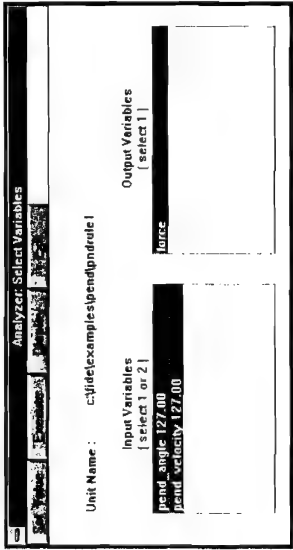


Figure 4-22. Analyzer/Select Variables Dialog Box

This dialog box allows you to select the input and output variables you wish to display. To select, click on the input variables and output variable of your choice. Alternatively you may use your keyboard to select the variables. Press the tab key to cursor to the variables list box, and then use the up or down arrow keys to select the variable of your choice.

Input variables and their values are displayed in the left list box, while output variables are displayed in the right list box. When selected, the entire line of the variable is highlighted. At most only two variables for input and one variable for output can be selected at one time. This constraint is imposed because only 3 dimensions (2 inputs & 1 output) can be easily visualized. However, different variables may be selected for different Analyzer sessions.

The dialog box contains a row of push buttons at the top of the window. These are used to further refine your tracing selections.

Set Input Value

Press the Set Value button to bring up the Set Input Value dialog box as shown in Figure 4-23.

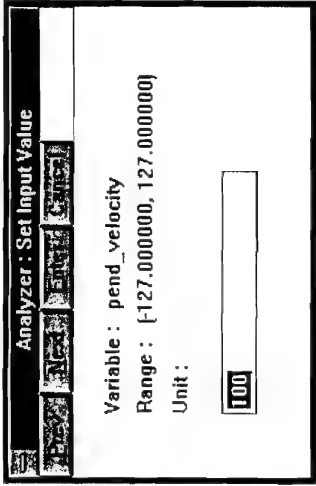


Figure 4-23. Analyzer/Set Input Value Dialog Box

This dialog box allows you to enter initial values for all input variables. Only those input variables not selected for display need be initialized. Selected variables are automatically assigned values spanning their entire domain. However, manually assigning selected variables has no ill effect on processing. Press the Next button or the Previous button to store the entered value into the selected variable and bring up the next or previous variable for initialization. Press the Cancel button to abort the current value entry. It does not abort values previously entered. Press the Enter button to store the entered value and return to the Select Variables dialog box.

All input variables must be initialized to valid values before Analysis can begin. The range of valid input values is displayed in the dialog box. Analyzer places default initial values in input variables when the Analyzer is first invoked. You should, however, override these entries with your own values, defining points of interest to you.

Execute

Press the Execute button to generate temporary data needed to display the different Analyzer views of your fuzzy inference unit. Before computation

begins a Select Precision dialog box comes up for your instruction. This Select Precision window is shown in Figure 4-24.

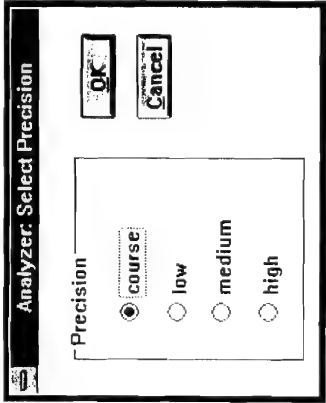


Figure 4-24 Analyzer/Select Precision Dialog Box

You have 4 choices:

- course
- low
- medium
- high

Precision determines the display resolution of Analyzer views. Course precision approximates data by skipping over computations for every 3 out of 4 points in a 256 x 256 grid. Low precision skips every 2 of 4 points, and medium precision skips every 1 of 4 points. High precision computes a values for each point in the grid. The trade-off is time versus resolution. A typical course precision computation takes 20 seconds. Contrast that to high precision, which can take up to 20 minutes.

Choose the precision by clicking one of the radio buttons and then pressing the OK button. An hour glass icon signifies Analyzer is unavailable for further work until precision directed computations are complete.

Display

Pressing Display initiates either one of two possible displays. (1) If you selected only one input variable to display a 2 dimensional (one input and one output) display is generated. (2) If you selected two input variables, a 3 dimensional display is generated.

Single Input Variable

Figure 4-25 shows an example of a single input display. All functionality of the Analyzer is available for the 2D display, except that you do not see the interaction with another input variable.

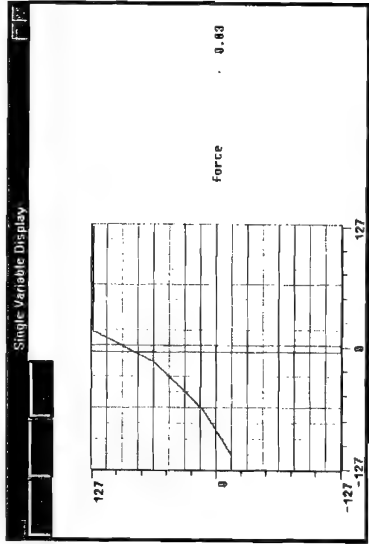


Figure 4-25 Analyzer 2 Dimensional Display

Move your cursor into the display region to activate the vertical hairline cursor.

As you move the cursor, note that the input value shown below the graph changes in response to your mouse position. The corresponding output value is displayed to the right of the graph. The buttons have the following functions:

- Trace** initiates the tracer function using values of input and output determined by the current position of the hairline cursor in the display.
- Exit** returns to the Analyzer/Select Variables display, where you can choose another variable or an additional variable.
- Grid** toggles the display grid on and off to enhance clarity if desired.

Two Input Variables

If you had selected 2 input variables for display, pressing the Display button brings up an Analyzer host window. 3 different Analyzer views appear in this Multiple Document Interface (MDI) window. Pull down the Views menu to inspect the choices available. The Analyzer window and its Views menu is shown in Figure 4-26. The 2 remaining menus of the Analyzer MDI are the Window and Help menus. They are identical in functionality to those found in the Fide window and described in chapter 1 for creating source files. The reader is referred to that chapter for information on the Window and Help menus.

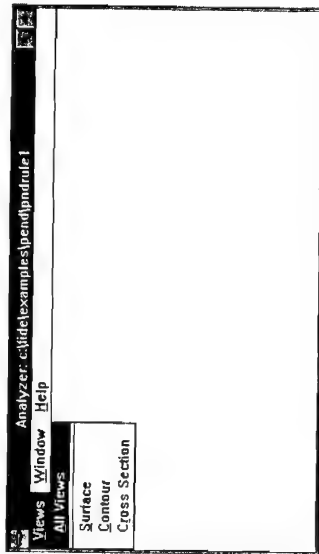


Figure 4-26 Analyzer Window

Analyzer Views

There are 4 menu commands.

All Views

The All Views command initiates automatic display of all 3 Analyzer views: Surface, Contour, and Cross Section. Each of these views is described below under the individual menu command for that view.

Surface

The Surface command launches a 3 dimensional display of your Fide inference unit transfer function. It is, of course, restricted to the 2 input and 1 output variables you selected prior to initiating this display. All other variables are assumed constant. A typical Surface view is shown in Figure 4-27.

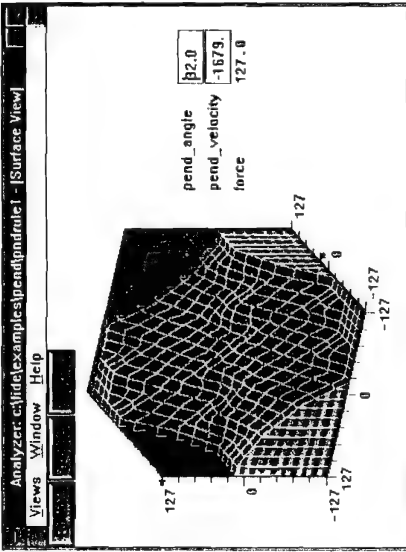


Figure 4-27 Analyzer/Surface View

Notice the 3 red arrows on each of the 3 axes of the surface view. The 2 horizontal axes represent the 2 input variables. The vertical axis represents the single output variable. Click on and drag one of the red arrows on the horizontal axes. Notice that the opposite input arrow remains constant while the output arrow moves in unison with your selected arrow. Notice also that the numeric values in text edit boxes to the right of the display follow your mouse movement as well. The arrow movement and numeric values represent positions on your 3 dimensional transfer function as you vary input across the domain of values. You can do the same thing (select red arrow and drag) for the second input variable as well.

To locate points precisely on your 3D surface, you may also enter numeric values for the 2 input points. Double click on a text edit box (located to the right of the display) to select the text, and type in a value. Press the tab key to register your value and move on to the next text edit box. The output value changes to reflect your new input points, and the arrow pointers move in step

with the revised values. This procedure is equivalent to dragging input arrows, except that it is precise - to the significant digits of your variables (see Fide Inference Language Reference Manual for information on defining variables).

To further investigate your Fide transfer function, 3 additional functions are provided with push buttons located at the top row of the window. They are Rotate, Trace, and Print.

Rotate Button

Press the Rotate button to bring up a dialog box allowing you to specify a new viewing angle and viewing pitch. This dialog box is shown in Figure 4-28.

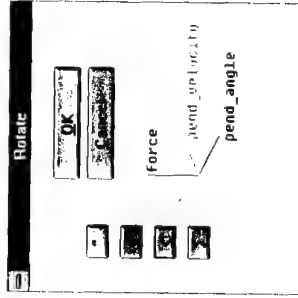


Figure 4-28 Surface View / Rotate Dialog Box

The 4 buttons on the left side of the window allow you to rotate the axes of the Surface view up, down, left (counter clockwise), and right (clockwise) respectively. A stick figure previews how the axes are positioned as you rotate the view. Press OK to initiate the rotation. The Surface view is entirely redrawn. Do not rotate frequently since each redraw can take up to a half a minute on a 386 class machine.

Trace Button

Press the Trace button to initiate a trace session using the values you have entered for the surface view, including those that are held constant. The trace procedure is identical to that described for the Tracer beginning in the Tracer Select Variables dialog box. Some of the initial steps, such as Set Input Values, of the Tracer procedure need not be repeated since they are performed in equivalent steps in the Analyzer/Surface View process.

Exit Button

Press the Exit Button to close the Surface View window. This button is equivalent to choosing the Close command from the window's control menu.

Contour

The Contour command launches a display of your fuzzy inference transfer function in true perspective. The Contour view is shown in Figure 4-29.

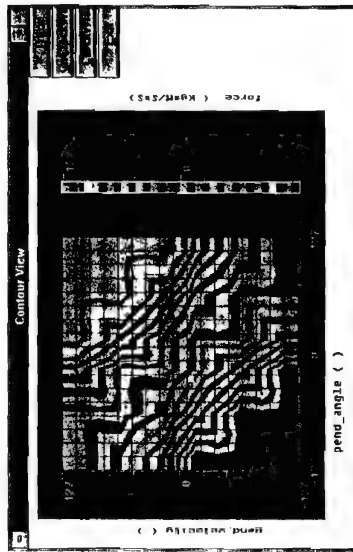


Figure 4-29 Analyzer/Contour View

The viewing angle is overhead looking straight down on the function surface (a top view). Color bands denote elevation levels over the surface, similar in effect to a geologic topographic map of terrain. An elevation bar to the right of the display acts as a scale and visual marker of output values as input points are selected in the contour region.

Click and drag within the contour region of the display. Note that cross hairs appear. They provide sight lines to input scales on the left and bottom edges of the display region. Also a red arrow tracks output along the vertical elevation bar. Furthermore, numeric values in text boxes below the display also track mouse movement, giving precision to point locations. You may also enter values in the text edit boxes to precisely position a point to the desired contour coordinate. Double click on the text edit box to select its text. Type in a numeric value, and then press the tab key to register the value and move on to the next text edit box. Note that the output value is recalculated each time you enter a new input value.

The Contour view provides 3 additional function buttons to assist you in analyzing your transfer function. They are the Trace, Rotate Counter Clockwise, and Rotate Clockwise buttons.

Trace

Press the Trace button to initiate a trace session using the values you have entered for the Contour view, including those that are held constant. The trace procedure is identical to that described for the Tracer beginning in the Tracer Select Variables section. Some of the initial steps, such as Set Input Variables, of the Tracer procedure need not be repeated since they are performed in equivalent steps in the Analyzer/Contour View process.

Rotate Counter Clockwise

Press the Rotate Counter Clockwise button to rotate the Contour View 90 degrees counter clockwise. The entire Contour View is redrawn.

Rotate Clockwise

Press the Rotate Clockwise button to rotate the Contour View 90 degrees clockwise. The entire Contour View is redrawn.

Exit Button

Press the Exit Button to close the Contour View window. This button is equivalent to choosing the Close command from the window's control menu.

Cross Section

The Cross Section command launches a display of two plane views of the transfer function cut across fixed points of input. This view is shown in Figure 4-30.

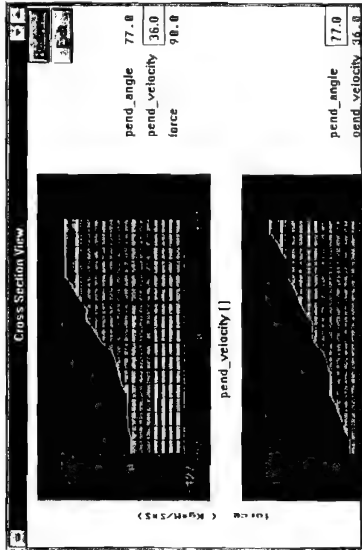


Figure 4-30 Analyzer/Cross Section View

These 2 views are in true perspective, with a viewing angle directly in front of the cross section cuts. Color bands are used to assist in visually calibrating elevation. Each view is perpendicular to the another, with each plane representing a fixed point on the opposite plane's input axis.

Click and drag your mouse within either of the two view regions. Two cross hair lines appear. This provides visual sight lines to the scales on the left and bottom boundaries of each region. Note that the horizontal cross hair is constrained to move along the outer boundary of the cut. This allows you to follow output as a function of input. The input and output values are also shown in text boxes to the right of the regions. These text boxes serve as data entry fields to allow precise positioning of the cross hairs. Double click on a text edit box to select it. Type in a new value, and then press the tab key to register the value and move on to the next text edit box. Note that the cross hair and output value track your changes.

Redraw Button

You can also examine different cross sections of your Fide inference transfer function. Position your input cursor (vertical cross hair) to a new point along the input scale (horizontal axis). Press the Redraw button. The Cross Section View is redrawn. Notice that the display region opposite the one on which you just moved the cursor has a new cut. This new cut is the plane view for the fixed input value represented by your repositioned cursor. This procedure is applicable to either region, but only one cut is updated for each redraw, the one whose input cursor is last repositioned. Two sessions are needed to change both cross section displays. This feature is intentional. Keeping one display region constant provides a stable reference for investigation of the opposite region.

Exit Button

Press the Exit button to close the Cross Section View window. Pressing Exit is equivalent to choosing the Close command from the window's control menu.

Simulator

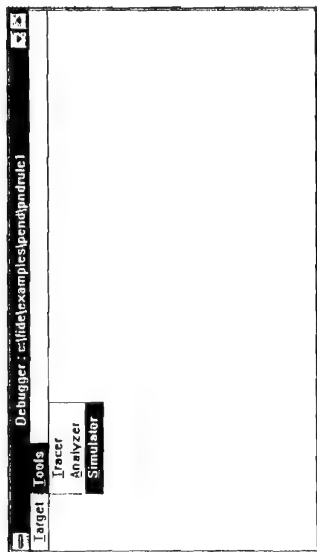


Figure 4-31 Simulator Command

The Fide Simulator, whose initiating command is shown in Figure 4-31, provides a software environment where your Fide Inference Unit can be viewed in the time domain. "Step" inputs are read into the Simulator from which a graph of outputs along with the inputs is plotted. Step is a user definable unit of time. The time varying curves of both input and output can be compared for convergence, discontinuities, and phase relationships.

Select Variables

Choose the Simulator command in the Debugger window. The Simulator/Select Variables Dialog Box comes up. This dialog box is shown in Figure 4-32.

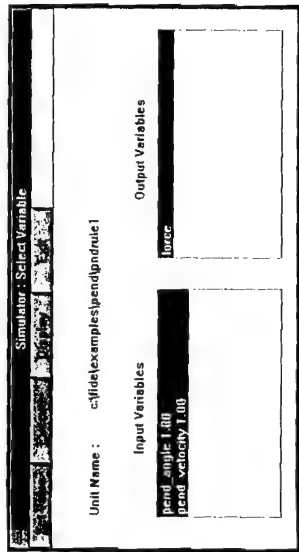


Figure 4-32 Simulator/Select Variables Dialog Box

This dialog box allows you to select input and output variables you wish to display. Click on the input variables and output variables of your choice.

Alternatively you may use your keyboard to select the variables. Press the tab key to cursor over to either the input or output list box, and then use the up or down arrow keys to select a variable. Press the return key to choose it.

Input variables and their values are displayed in the left list box. Output variables are displayed in the right list box. When selected, the entire line comprising the variable is highlighted. At most only 7 input variables and 7 output variables can be selected. However, a different set of 14 may be selected for different Simulator sessions. Note that input variables have values displayed. These are default values to be held constant while selected variables vary in the simulation. If a variable is selected (highlighted), the default value is not used. Input values, for each step of the simulation, will be read in from the Input file. Output variables have no default values, as they are a function of inputs.

The dialog box contains a row of push buttons at the top of the window. These are used to further refine and display your variables.

Set Input Value

Press the Set Value button to bring up the Set Input Value dialog box as shown in Figure 4-33

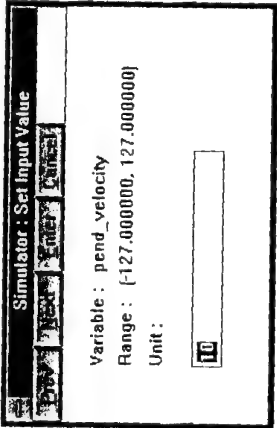


Figure 4-33 Simulator/Set Input Value Dialog Box

This dialog box allows you to enter initial values for all input variables. Only those input variables not selected for display need be initialized. Selected variables are automatically assigned values spanning their entire domain. However, manually assigning selected variables has no ill effect on processing. Press the Next button or the Previous button to store the entered value into the selected variable and bring up the next or previous variable for initialization. Press the Cancel button to abort the current value entry. It does not abort values previously entered. Press the Enter button to store the entered value and return to the Select Variables dialog box.

All input variables must be initialized to valid values before Simulator can begin. The range of valid input values is displayed in the dialog box. Simulator places default initial values in input variables when the Simulator is first invoked. You should, however, override these entries with your own values, defining points of interest to you.

Execute

Press the Execute button to generate temporary data needed to display the Simulation. An hour glass icon is displayed for the duration of the Execution. Execute must be performed before Simulation starts. Otherwise a previously created temporary data file will be incorrectly used, or, if no temporary data file is available, an error message is displayed.

Display

Press the Display button to launch your Simulation display. A typical Simulation/Display window is shown in Figure 4-34.

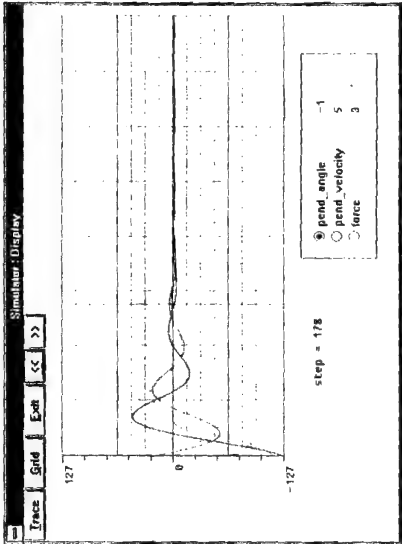


Figure 4-34 Simulation/Display Window

The Simulation display region consist of a graph of inputs and outputs varying across the step domain. A step is a user definable duration of time (see

Appendix B Fide input file converter for more information). Displayed below the graph are the variable names and their values.

Move your cursor across the region of the display where curves are plotted. As you do so, a vertical hair line appears to provide a visual point of reference. Note that step values and listed variable values track your cursor movement. These numeric figures give precise position information. Select a variable by clicking on its radio button. The vertical scale on the left boundary of the graph changes to match your variable selection. The scales are color matched with curves in the graph for easy identification.

A series of push buttons at the top of the window provide additional functionality. They are described below.

Trace Button

Trace allows you to utilize the Tracer function of Fide to identify source statements responsible for Simulation curves. Trace automatically takes you into the Select Variable window of Tracer. This window and a trailing series of dialog boxes is described beginning in Tracer Select Variables section of this chapter. Initial variable values and the Fide inference unit needed for Tracer are obtained from the Simulator. Once in Tracer the procedure for tracing to a specific source statement is the same as for general use of Tracer in Fide.

Grid Button

Press the Grid button to toggle on and off the display of grid lines on the Simulation display. There are times when curves are so densely packed together that the disabling of background grid lines add to clarity.

Exit Button

Press Exit to close the Simulation window. Exit is equivalent to choosing the Close command from your window control menu.

<< Button

Press the "<<" button to load into memory an additional 512 steps of Simulation display. The Simulator display can only display up to 512 steps. If simulations are longer than 512 steps, Fide buffers the non-visible portions of the display on disk.

>> Button

Press the ">>" button to load into memory a prior 512 steps of Simulation display. The Simulator can only display up to 512 steps at one time. If simulations are longer than 512 steps, Fide buffers the non-visible portions of the display on disk.

< Button

Press the "<" button to move the cursor (vertical hair line) a single step to the left (decrement step by 1). This feature is provided to more easily position the step count precisely.

> Button

Press the ">" button to move the cursor (vertical hair line) a single step to the right (increment step by 1). This feature is provided to more easily position the step count precisely.

Chapter 5 Generating Run Time Code

Fide provides direct compatibility with off-the-shelf controller chips manufactured by Motorola. This is accomplished by converting Fide compiler object output into standard Motorola MCU's assembly code. Currently Fide supports three Motorola chips. They are:

- MC6805
- MC68HC05
- MC68HC11

The following sections describe the Fide commands and actions needed to generate run time code.

RTC Menu Commands

RTC is an acronym for Real Time Code. RTC converts Fide compiler object output into assembly code for specific Motorola chips. The RTC menu is shown in Figure 5-1.

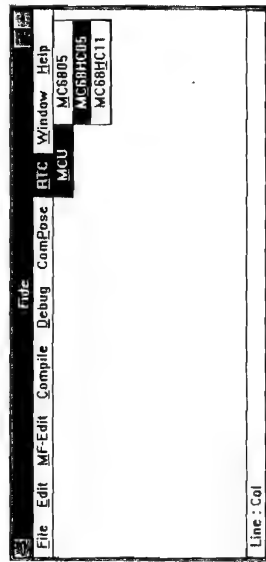


Figure 5-1 Fide/RTC Menu

A source file must be open and compiled before Real Time Code can be generated. RTC looks for the compiled object of the open file, from it converts

to assembly code. When conversion processing finishes, a message box displaying the phrase "RTC Convert Success" comes up as shown in Figure 5-2.



Figure 5-2 Fide/RTC/Conversion Success Message Box

A Messages Window also comes up immediately below the source file edit window. This Messages Window, identical to the one which comes up after Fide source compilation, lists error messages, if any, of the conversion process (see Figure 3-3). Unlike source compiler messages, converter messages cannot be traced back to a particular source statement. It's input is not source, but rather object output of source.

Motorola Chip Support

RTC currently supports 3 Motorola MCU chips. They are described in the following sections.

MC6805

Resolution of I/O Data:
8 bit
Resolution of Grade Value:
Maximum of 255
Number of Input Variables:
Maximum of 255
Number of Output Variables:
Maximum of 16 labels/variable
Membership Function Shape:
Any kind of polygon with less than 6 edges
Number of Rules:
Memory size dependent
TVFI
Inference Method
Center of Gravity, Left Maximum, Right Maximum, and Maximum Average.
Defuzzification:
Fide Output File:
.xxxxx.asm
where xxxxxx is the rule source file name

MC68HC05

Resolution of I/O Data:
8 bit
Resolution of Grade Value:
Maximum of 255
Number of Input Variables:
Maximum of 255
Number of Output Variables:
Maximum of 16 labels/variable
Membership Function Shape:
Any kind of polygon with less than 6 edges
Number of Rules:
Memory size dependent
TVFI
Inference Method:
Center of Gravity, Left Maximum, Right Maximum, and Maximum Average
Defuzzification:
Fide Output File:
.xxxxx.asm
where xxxxxx is the rule source file name

MC68HC11

- Resolution of I/O Data:
 - Resolution of Grade Value:
 - Number of Input Variables:
 - Number of Output Variables:
 - Number of Labels
 - Membership Function Shape:
 - Number of Rules:
 - Inference Method:
 - Defuzzification:
 - Fide Output File:
- 8 bit
- 8 bit
- Maximum of 255
- Maximum of 255
- Maximum of 16 labels/variable
- Any kind of polygon with less than 6 edges
- Memory size dependent
- TVFI
- Center of Gravity, Left Maximum, Right Maximum, and Maximum Average
- xxxxxx.asm
- where xxxxxx is the rule source file name

Composer is the linking function of Fide. Individual Fide Inference Units, Fide Operation Units, and Fide Execution Units are connected into a unified whole using Composer tools. The Composer also supports features to analyze and debug your system. Once debugged, C libraries and executable code can be generated to represent the logic of your fuzzy system. The Fide command to initiate Composer is shown in Figure 6-1.

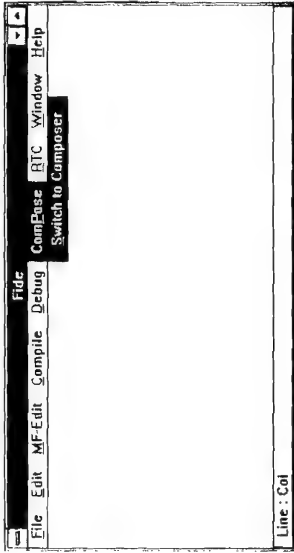


Figure 6-1 Fide/Composer Command

The Fide Composer addresses system level requirements of the fuzzy applications developer. Once you have designed, compiled, debugged, tested, and tuned your Fide inference units, Fide allows you to link individual Fide inference units, along with other types of units, into an integrated whole. As an example, say you wanted to create a cascaded 2 stage inverted pendulum control system. You use the Fide Compiler and Debugger to develop an inference unit for a single stage control. You then use the Fide Composer to link two of these units in tandem (cascaded).

Composer supports a rich set of tools to create, debug, tune, and utilize fuzzy inference systems. You can create a system from a blank work space, or you can build on existing systems by adding, deleting, or modifying component units. Composer supports and creates systems as graphical entities or equivalent text entities. Both forms are stored as ASCII text in DOS files for easy viewing and

debugging, and can be readily converted from one form to the other. From a Fide system file, Composer can generate a variety of useful code files, including ANSI C source (for porting to standalone C language applications), PC executable code, simulation data files, and data flow files. These features comprise a complete and robust development environment for systems development.

Composer Functions

Upon choosing the Switch To Composer command in the main menu of your Fide window, the Fide Composer window, which is shown in Figure 6-2, comes up.

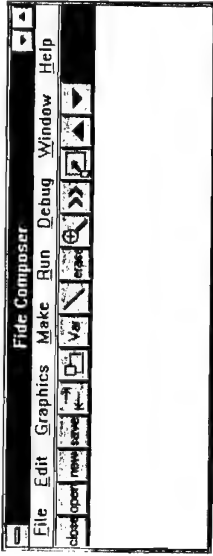


Figure 6-2 *Composer Window*

Fide Composer can be thought of as a separate application within the Fide development environment. Its window has its own menu bar whose commands operate on a different type of object than that of Fide. Composer acts on a System object (linked and integrated Fide units) rather than on a single unit object. Fide focuses on a Fide unit object (rules and definitions of a single inference unit).

The commands of Composer are described in the sections below. Note that for some commands, Composer includes equivalent visual push buttons at the top row of the window. These are tool buttons to improve access to the most

frequently used functions. Also all source files of Fide units which are to be used in the Composer must be compiled before they can be so used.

File

The File menu manages files representing your Fide System application. This menu is shown in Figure 6-3.

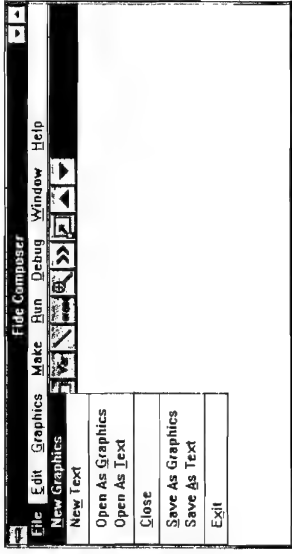


Figure 6-3 *Composer/File Menu*

The File commands are similar to file functions in other parts of Fide. However, they are enhanced to facilitate handling Composer files.

New Graphics

The New Graphics command allows you to create a new system in graphics format. A blank untitled window appears showing only a skeleton grid. The drawing functions including "draw unit" are explained in later sections of this chapter. The New Graphics window is shown in Figure 6-4.

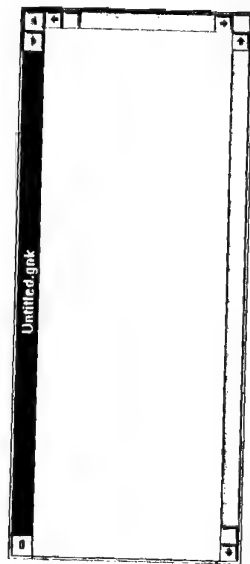


Figure 6-4 Composer/New Graphics Window

New Text

The New Text command is the complimentary command to New Graphics. It allows you to create a new system in text format. A blank untitled text edit window appears. The cursor changes into a text insertion bar, indicating typing can begin here. Data entry is free form text with full screen edit capabilities identical to those of the Fide Source text editor. Edit functions are described in chapter 1 under the sections Edit Menu Commands and Keyboard and Mouse Editing Functions. Syntax rules for Composer text files are explained in the Fide Composer Language (FCL) section of the Fide Reference Manual. A typical Composer Text File window is shown in Figure 6-5.

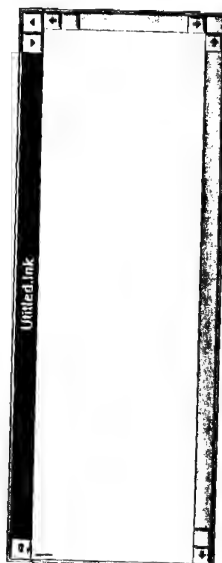


Figure 6-5 Composer/New Text

Open As Graphics

The Open As Graphics command opens an existing Fide system which had been previously saved as a graphics or text format file. A dialog box comes up allowing you to select the file to open. This dialog box is shown in Figure 6-6.

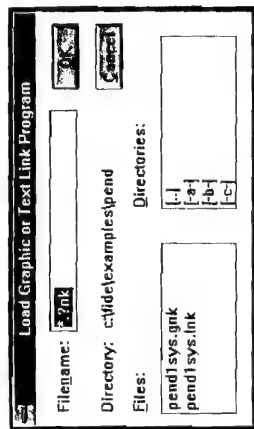


Figure 6-6 Composer/Open As Graphics Dialog Box

The dialog box conforms in style to other "open file" dialog boxes in Fide. However, the file mask (filter) for listing files is initially set to *.*nk. Only files having the graphics (*.gnk) or text (*.lnk) format extension are allowed to be opened. The filter may be modified to list other files in the directory. However, only those files with the extensions ".gnk" or ".lnk" will be opened. If you select a Composer Text format file, Open As Graphics automatically converts the file into Graphics format. The new file name is the text file name with an extension string of ".gnk". A window appears containing existing data displayed in the graphics format. A typical Composer window in graphics format is shown in Figure 6-7.

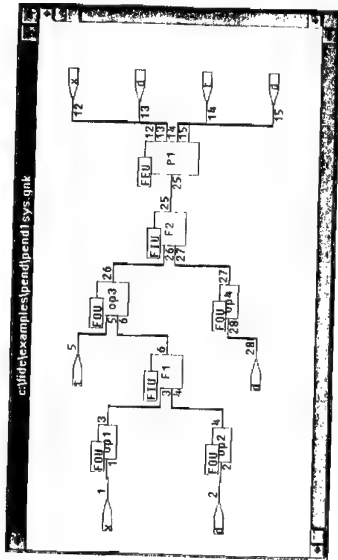


Figure 6-7 *Composer/Graphics File Window*

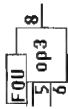
A Composer graphics window contains a number of components comprising a complete Fide system. These components are introduced below.

Fide Inference Unit (FIU)



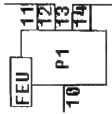
The Fide Inference Unit is the "embedded" controller component of a Composer system. Other components support FIUs, or provide data to or take data out of FIUs. The FIU must first be compiled under Fide before using in Composer. Composer looks for the FIU's map file.

Fide Operation Unit (FOU)



The FOU provides interfacing between separate FIUs. FIUs are instruction modules that map data from one form to another so that FIUs can remain independent of one another and yet be used together. FOU must be created before entering the Composer. FOU instructions are specified in the Fide Inference Language (FIL), and compiled like regular Fide source files. The map file outputs are accessed by Composer as FIUs.

Fide Execution Unit (FEU)



The FEU provides interfacing to external object modules. An FEU points to an externally compiled object file. Currently, Composer support Turbo C or Borland C object files. Object file name, input and output variables, and other parameters are specified using the Fide Inference Language (FIL). The FEU source is compiled using the Fide Compiler. Composer searches for the map file generated by the compiler.

FEUs can be thought of as the application (e.g. plant) in which FIU controllers are embedded. The object modules themselves are not included in the Fide system, but references to them are.

Fide Input Node



The Input Node signifies a point of data entry for a Fide inference system. The values going into a node are an initial excitation value (specified when the node is created) and subsequent generated values, usually through feedback from output nodes. Create an Input Node using Graphics menu tools.

Fide Output Node



The Output Node signifies a point of data egress or exit from the Fide inference system. Values coming out of a node can be fed back into an input node effecting a closed loop control. This is accomplished by specifying the same identifier for both an output node and an input node. Create an Output Node using Graphics menu tools.

Open As Text

The Open As Text command opens an existing Fide system file previously created or saved in text format. A file open dialog box comes up allowing you to select a ".lnk" or ".gnk" file to open. The file selection dialog box is shown in Figure 6-8.

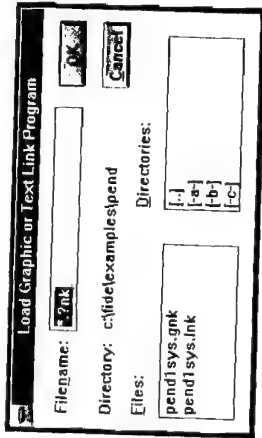


Figure 6-8 Composer/Open As Text Dialog Box

The Open As Text dialog box functions similar to most Open File dialogs. However, this dialog box filters files with an *.lnk file mask so that only Composer files can be opened. You can modify this filter to list other files in directory, but only files with the extensions ".gnk" or ".lnk" can be opened. If you select a Composer Graphics file, Open As Text automatically converts it into Text format. The new file name is the same file name with an extension string of ".lnk". Upon opening a text file, a window appears containing existing data in the text format. A Composer text window is shown in Figure 6-9.

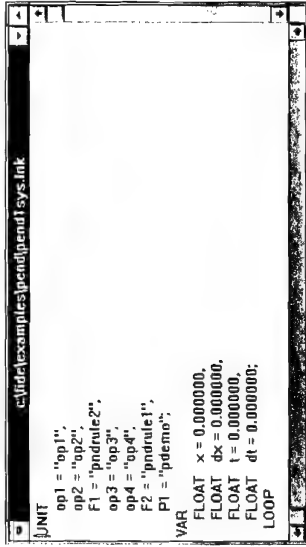


Figure 6-9 Composer/Text File Window

The Composer Text File is a freeform text document, editable by virtually any text editor including Fide's own text editor. The file is organized into distinct blocks of text segregated by function. There are blocks for unit definitions, variable definitions, loop definitions, and others. The syntax and usage of Fide Composer Language is explained in the Fide Composer Language (FCL) section of the Fide Reference manual.

Close

The Close command closes the active window in Composer. It is equivalent to the Close command of the Control menu for that window or the keyboard combination of CTRL-F4.

Save As Graphics

The Save As Graphics command stores your Fide inference system in graphics format. A dialog box appears requesting name and path you wish to assign to your saved system. This dialog box is shown in Figure 6-10.

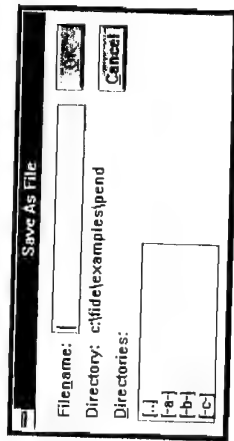


Figure 6-10 *Composer/Save As Graphics Dialog Box*

Note that you may use any file name and store it in any path. However, Composer will always append a ".gnk" extension to the name. gnk is the extension Composer looks for when you next try to open an existing graphics file.

The Save As Graphics command can perform automatic conversion of Composer files from text form to graphics form. If the active document is a text Composer file, you can convert this file into a graphics Composer file and save it by using the Save As Graphics command. Composer places a ".gnk" extension to whatever name you enter.

Save As Text

The Save As Text command stores your Fide inference system in text format. A dialog box appears requesting the name and path you wish to assign to your saved system file. This dialog box is the same as for the Save As Graphics dialog box shown in Figure 6-10.

Note that you may use any file name and store it in any path. However, Composer will always append a ".lnk" extension to the name. lnk is the extension Composer looks for when you next try to open an existing Composer text file.

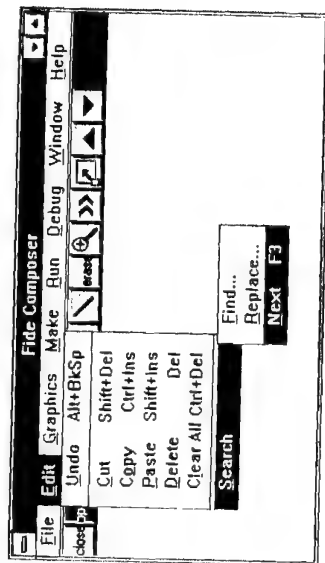
The Save As Text command can perform automatic conversion of Composer files from graphics form to text form. If the active document is a graphics Composer file, you can convert this file into Composer text and save it by using the Save As Text command. Composer places a ".lnk" extension to whatever name you enter.

Exit

The Exit command closes the Composer window. It is equivalent to the Close command in the System Control Menu. If you have not saved the file on which you are working, Composer issues a warning message prompting you to close without saving, or to cancel the Exit command.

Edit

The Edit menu is intended for use with Composer Text files. It has no effect on Composer Graphics files. The functions are standard text editing functions available throughout Fide. The Edit menu is shown in Figure 6-11.

Figure 6-11 *Composer/Edit Menu*

The Edit menu commands are described in the sections below.

Undo

The Undo command cancels your previous edit command and restores text to its former state. For example, if you inadvertently deleted a selected phrase, you can restore the deleted characters by choosing Undo. One level of Undo is supported. Only the most recent edit command can be undone.

Cut

The Cut command copies a selected string of characters into the Windows clipboard and deletes it from the file image being edited. See chapter 1, Keyboard and Mouse Editing Functions, for details of how to select and move around in text using the mouse and keyboard.

Copy

The Copy command copies a selected string of characters from the edit window to the Windows clipboard. The original text stays intact unchanged in the edit window.

Paste

The Paste command inserts the current contents of the Windows clipboard into the active edit window at the current cursor insertion position. If, at the time the Paste command is chosen, a string of characters is selected in the edit window, it will be overwritten by the Paste text.

Clear All

The Clear All command blanks out all text in the edit window. This command allows you to easily start over. However, the original file is not changed until you choose the Save As command.

Search

The Search ... command brings up an hierarchical submenu containing 3 items:

- Find ...
- Replace ...
- Next

Find

The Find ... command brings up a dialog box in which you are asked to enter a find string. This dialog box is shown in Figure 6-12.

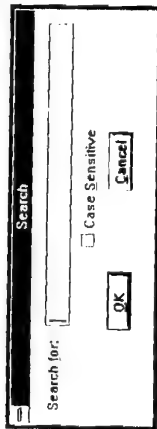


Figure 6-12 Edit/Find Dialog Box

Replace

The Replace ... command brings up a dialog box in which you are asked to enter both a search string and a replacement string. This dialog box is shown in Figure 6-13.

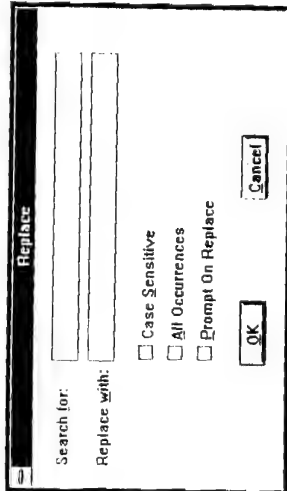


Figure 6-13 Edit/Replace Dialog Box

Pressing the OK button in either the Find ... or Replace ... dialog boxes positions the cursor to select an occurrence of the search string in the edit window. In the case of Replace, note that you can choose options to either globally replace or be prompted for replacement at every found instance.

Next

The specified string search can continue throughout the document by pressing the F3 key or choosing the Next command of the Search menu.

Graphics

The Graphics menu provides the tools to create and maintain a Fide inference system. An inference system is represented in Composer by a schematic diagram. You use Composer tools to draw the schematic of your Fide application, using as components pre-developed inference units and C object modules. A Composer Graphics file stores the components and links of the schematic. The Graphics menu is shown in Figure 6-14.

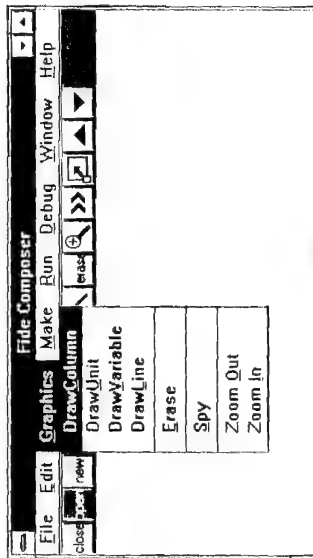


Figure 6-14 Composer/Graphics Menu

The Graphics menu commands are described in the sections below.

Draw Column



The Draw Column tool allows you to insert blank columns into a Fide inference system schematic. Additional components can be added in the new space. You can use this tool to add units in series or in parallel.

To add a blank column, choose the Draw Column tool. The cursor icon changes to a draw column icon. Double click on the column into which you wish to insert a new column. The entire screen redraws, and existing columns are shifted to the right of the new blank column. You can now choose another tool to add, delete, or modify components of your schematic. A Composer schematic with a newly added column is shown in Figure 6-15.

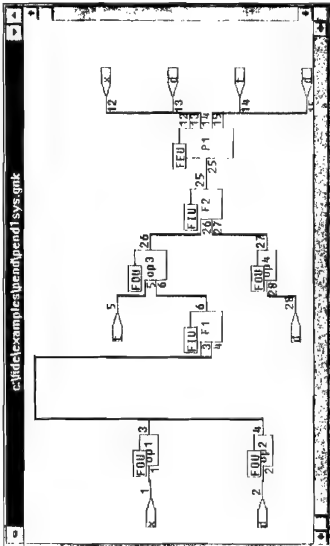


Figure 6-15 Composer Graphics File with New Column

Draw Unit



The Draw Unit tool allows you to add either a Fide Inference Unit (FIU), Fide Operation Unit (FOU), or Fide Execution Unit (FEU) to your Fide system schematic. The units must have already been compiled using the Fide Compiler and the Fide Inference Language (FIL). The Draw Unit tool selects a unit and identifies it to Composer.

To add a unit, position the cursor in your Composer schematic diagram. Double click the Draw Unit tool at this location. A dialog box comes up prompting you to select a map file and to enter a name for the unit. The dialog box is shown in Figure 6-16.

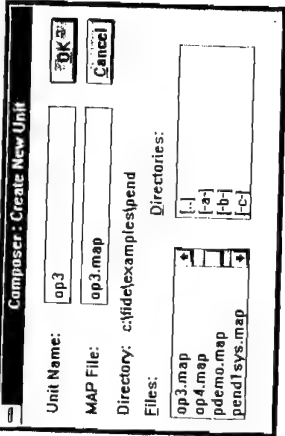


Figure 6-16 Composer/Draw Unit Dialog Box

Select the unit map file by clicking on the file name in the list box. Do not double click on the name. Type in the display name in the text edit box at the top of the dialog box. You may use any name. However, it is recommended the display name be the same as the file name. Press OK to insert the unit. Composer will rearrange the existing units to allow the new unit to fit in. The Composer schematic display is redrawn. Note that the new unit is still isolated

from other elements of your Fide system. You must connect it to other components using the Draw Line tool (see section 6.3.4 below).

Draw Variable



The Draw Variable tool allows you to define and direct data flow in your Fide system schematic. There are 2 data nodes you may define. They are:

1. input - identifies a point where data flows into the system.
2. output - identifies a point where data flows out of the system.

To add a variable node, select the Draw Variable tool and position the cursor in your Composer schematic diagram. Double click. A dialog box appears prompting you for the type of node and the identifying name. The dialog box is shown in Figure 6-17.

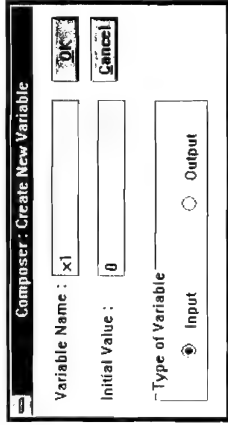


Figure 6-17 Composer/Draw Variable Dialog Box

Type in a name for the node in the text edit box provided. Select the type of node you want by choosing one of the radio buttons. Note that if you enter an output node name that is identical to an input node name, or vice versa, you are establishing a feedback loop in your schematic. The Initial Value text edit box is blank when you first enter the dialog box. However, you may enter a value

override the default value of zero. Press OK to confirm. Composer redraws the diagram to insert your new node. Your node is still isolated (unconnected) to other components. You must connect it to the system using the Draw Line tool (see below).

Draw Line



The Draw Line tool allows you to connect two components of the system together. It performs two functions: (1) identifies which outputs of a source element goes to which inputs of a target element, and (2) draws a line to visually connect the two components together.

To connect two Fide system components together, select the Draw Line tool and double click over the source element. A dialog box appears prompting you to select the output you want to connect. The dialog box is shown in Figure 6-18.

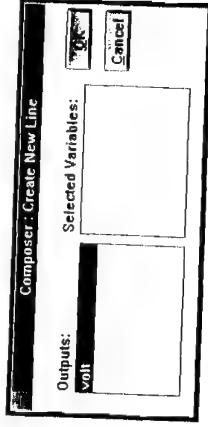


Figure 6-18 Composer/Draw Line Source Node Dialog Box

There are 2 list boxes. The one on the left is a list of as yet unselected outputs. The list box on the right contains outputs that have already been taken. Double click on an item in the left list box, or alternatively select (highlight) a name and press the OK button. A pin at the source element turns blue to identify the selected output.

To select the target or destination element, double click over the intended unit or output variable. A dialog box appears as shown in Figure 6-19.

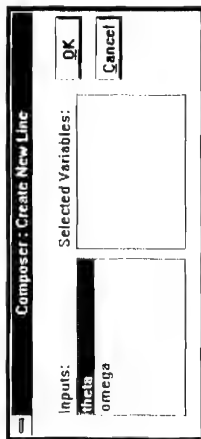


Figure 6-19 *Composer/Draw Line Target Node Dialog Box*

As with the source element dialog box, there are two list boxes. The one on the left identifies inputs available for selection. The one on the right shows inputs that are already taken. Double click on an input in the left list box, or select (highlight) a name and press the OK button.

Composer redraws your schematic diagram with a line connecting the output and input points you selected. Note that a unique number has been assigned to the output and input points. This number identifies your connection uniquely. To minimize clutter, a single undifferentiated "bus" line is used to represent connections of the system across column boundaries. A new connection resulting from the user instructions is shown in Figure 6-20.



Figure 6-20 *Composer/New Line Connecting Components*

Erase



The Erase tool allows you to delete components from your Fide system schematic. The symbols representing units, variables, lines, and columns can be deleted. The underlying files represented by the symbols, though, are not deleted.

To erase, select the Erase tool. The cursor changes into the erase icon. Place the erase tool on top of the item you wish to erase. Double click. A confirmation dialog box appears. This dialog box is shown in Figure 6-21.



Figure 6-21 *Composer/Erase Confirmation Dialog Box*

Press OK. Composer redraws the schematic diagram display without the deleted component.

Spy



The Spy tool allows you to examine the details of any visual component of your Fide system schematic. You can display the map file name, the type of component it is, and list the inputs and outputs.

To Spy on a component, select the Spy tool. The cursor changes to the icon. Position Spy on top of the item in which you are interested. **Do** A dialog box with information pertaining to the selected item appears. The dialog box is shown in Figure 6-22.

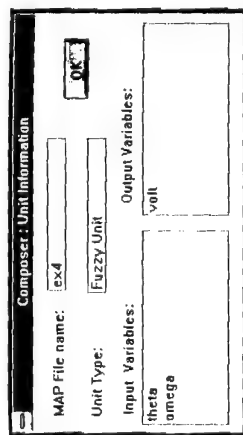


Figure 6-22 Composer/Spy Tool Dialog Box

Zoom Out

The Zoom Out command causes your Fide schematic diagram to become smaller or magnified. Choosing this command repeatedly will continue to increase the size of the diagram relative to the display. It is from the perspective of the schematic, which is expanding out.

Zoom In

The Zoom In command causes your Fide schematic diagram to become smaller or demagnified. Choosing this command repeatedly will continue to decrease the size of the diagram relative to the display. It is from the perspective of the schematic, which is shrinking in.

Make

The Make menu instructs Composer to generate various files which are used both inside and outside of Fide. Outside files provide a means of porting fuzzy inference work to other platforms and applications. Inside files provide application specific intermediate data for display processing in the debugging and analysis phases of development in Composer. The Make menu is shown in Figure 6-23.

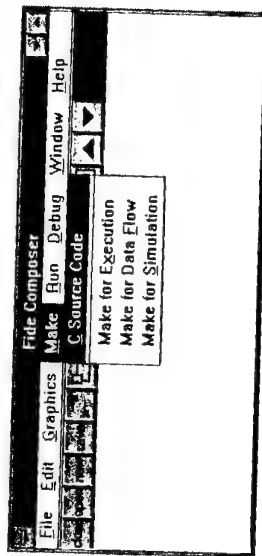


Figure 6-23 Composer/Make Menu

Each of the Make menu commands are described below.

C Source Code

The C Source Code command initiates generation of ANSI C statements representing the full functionality of your Fide inference system. Fide Operation Units (FOU) and Fide Variables are translated directly into C source code.

Fide Inference Units (FIUs) are converted into C source as well. However, the C code generated are C function calls (references) to the Aptrox Runtime Library. The Runtime library is linked into your resultant executable file.

Fide Execution Units (FEU) are also converted to C source. However, FEU references to C object modules developed independent of Fide. Similar to the translation of FEUs into C code consist of function calls to external C routines whose object code is linked into the executable program. The FEU code generated by Fide does not include the external C code referred to by FEU.

If the Make into C Source Code is successful, a new text window showing generated C code appears. You can examine and edit this code as you can any other text edit file in Fide. An example C Source Code window is shown in Figure 6-24. If the Make command is not successful, an error messages window appears. Error messages are listed in a scroll list box within the window so you may easily access each message.

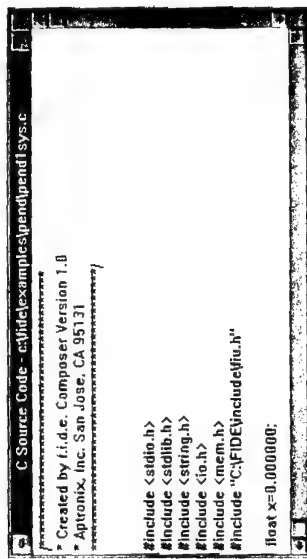


Figure 6-24 *Composer/Make C Source Code Window*
Make For Execution

The Make For Execution command compiles the C source code generated by Composer. Composer executes a command line string to perform the compilation. Specifications of the resident compiler as well as the memory model are defined in the file "FIDEMAKE.CFG". The default compiler is Turbo C using the large memory model. Support for other C compilers is also

available. However, you must modify FIDEMAKE.CFG, which consists of two ASCII text lines, each of which is a single word. The supported options are:

second line	resident compiler
tec	Turbo C
bcc	Borland C
cl	Microsoft C 6.0
second line	memory model
T	tiny
S	small
M	medium
C	compact
L	large
H	huge

Once the Make For Execution command has been invoked and the command string initiated, a DOS message window appears tracking the progress of the compilation and linking process. The messages are in DOS text mode, and come directly from the compile and link steps of your FIDEMAKE.CFG specified compiler program. A typical DOS message window is shown in Figure 6-25.



Figure 6-25 *Composer/C Compiler Message Window*

The result of Make For Execution is an *exe* file, executable on any PC hardware platform.

Make For Data Flow

The Make For Data Flow command generates an executable file similar to that of the Make For Execution command. The difference is that Make For Data Flow includes additional data output. The extra data is used to drive the display of information in the Debug/Data Flow command discussed later in this chapter.

Once the Make For Data Flow command has been invoked and the command string initiated, a DOS message window appears tracking the progress of the compilation and linking process. The messages are in DOS text mode, and come directly from the compile and link steps of your FIDEMAKE.CFG specified compiler program. A typical DOS message window is shown in Figure 6-25 above.

The result of the Make for Data Flow command is an *exe* file, executable by the Composer's Run command (see below).

Make For Simulation

The Make For Simulation command generates an executable file similar to that of the Make For Execution command. The difference is that Make For Simulation includes additional data output. The extra data is used to drive the display of information in the Debug/Simulation command discussed later in this chapter.

Once the Make For Simulation command has been invoked and the command string initiated, a DOS message window appears tracking the progress of the compilation and linking process. The messages are in DOS text mode, and come directly from the compile and link steps of your FIDEMAKE.CFG specified compiler program. A typical DOS message window is shown in Figure 6-25 above.

The result of the Make for Simulation command is an *exe* file, executable by the Composer's Run command (see below).

Run

The Run menu provides options for creating temporary data files needed for performing data flow and simulation displays of your Fide inference system. The Run commands do not themselves display any information. They set up the conditions and generate the internal data for these displays. The Run menu is shown in Figure 6-26.

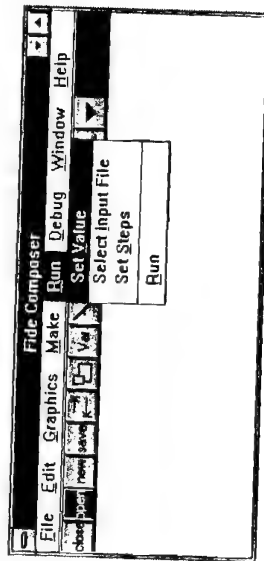


Figure 6-26 Composer/Run Menu

Set Value

The Set Value command allows you to override default values for input variables of your system. When you create input variable nodes, initial values are entered at that time (see Figure 6-17). Those are the default values if you choose not to enter different values in this dialog box. This command makes it convenient to change values for purposes of analysis.

Choose the Set Value command. A dialog box comes up listing the variables belonging to your system. This dialog box is shown in Figure 6-27.

value and to return to the Set Value dialog box. Press the Cancel button to abort any changes you made to the value and to return to the Set Value dialog box.

Select Input File

The Select Input File command allows you identify an input file for use in Composer's debugging displays. Specification of a series of input values for all variables for every step interval of the data flow and simulation is needed by Composer to generate data for display. These values are stored in a file with the extension name of SIN. The format and syntax of the Input File is explained in the Appendix of the Fide Composer Language section of the Fide Reference Manual.

Choose the Select Input File command. A Select Input File dialog box comes up for your review. This dialog box is very similar to the an Open File dialog box. You can interact with it as you would an Open File dialog box. The dialog box is shown in Figure 6-29.

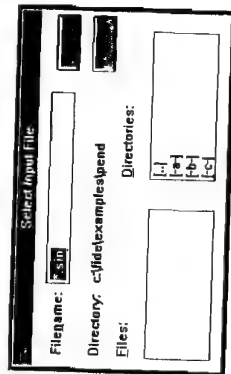


Figure 6-29 Composer/Select Input File Dialog Box

A file filter of "*sin" displays files with a SIN extension name in the left file list box. Double click on the file you wish to use. The right list box allows you to change directories if the input file is not in the current default directory. Alternatively, you may type in an explicit file name, overriding the SIN extension. Press the OK button to register the entered file name with Composer.

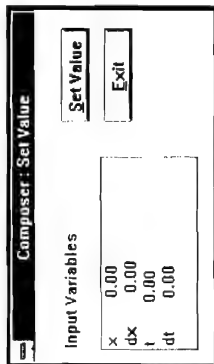


Figure 6-27 Composer/Set Value Dialog Box

A scrollable list box contains all the variables of your Fide system. Choose a specific variable by double clicking on it or highlighting it (single click on it) and pressing the Set Value button. You must do this with every variable for which you wish to change a value.

Upon choosing a variable, a second dialog box comes up for you to modify a variable value. This dialog box is shown in Figure 6-28.

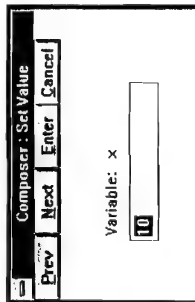


Figure 6-28 Composer/Enter Variable Value Dialog Box

To enter or change a value, double click in the text edit box to select the value. Type in a numeric value within the range allowed for that variable. Press the Next or Prev button to register the value with Composer and to go on to the next or previous variable. Alternatively, Press the Enter button to register the

The Input File is used for both *open* loop and *closed* loop systems. In an *open* loop system, there is no feedback from output back to input. In this situation, every line of the input file is used as input values (in successive steps). In contrast to this, a closed loop system provides feedback from output to input; thus generates its own input over time. In this situation, only the first line of Input File is used.

Set Steps

The Set Steps command allows you to determine the number of iterations for the Data Flow and display and the number of time intervals for the Simulation display. They both use the same number. A step is a single complete loop through the system. The Simulation display shows only values at the entry and exit of the loop. The Data Flow display shows intermediate values of stages within the loop as well. The default step value is 1,000. This is the number of data groups (one groups per loop) generated by the Run command (see below).

Choose the Set Steps command. A dialog box prompting for the number of steps appears. This dialog box is shown in Figure 6-30.



Figure 6-30 Composer/Set Steps Dialog Box

Type in the number of steps in the text edit box and press the OK button. The larger the number, the larger the temporary data file generated for display.

Run

The Run command initiates generation of temporary data files for Data Flow and Simulation displays. Composer runs the executable file created by the Make For Data Flow or Make For Simulation commands of the previous menu. The *exe*

programs so run create the temporary debug data files. Executing of this command changes the mouse cursor to an hour glass to indicate work in progress. Composer is unavailable to perform other work while this hour glass icon remains in effect.

Note that Run does not initiate a display (unless the *exe* program described above includes a display component as part of its data generation routine). Only commands under the Debug menu invoke Composer originated displays. Run prepares the data for the number of specified steps needed by Debug to create displays representing data flow. Also only the data file for the Make option selected under Make is created. If Make For Data Flow was selected, the resulting Run data file is only good for the Debug/Data Flow display. If Make For Simulation was selected, the Run generated data file is only good for the Debug/Simulation display.

Debug

The Debug menu initiates the tools to help you analyze, detect errors, and correct problems in your Fide Inference System. The Debug menu is shown in Figure 6-31.

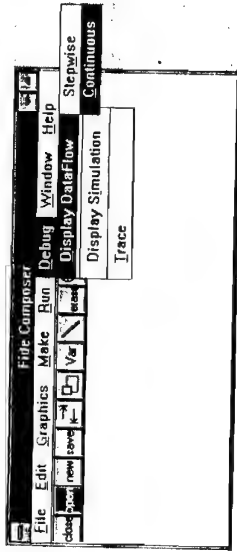


Figure 6-31 Composer/Debug Menu

Debug commands provide information on your Fide application as a whole. If you need to investigate individual components of the system, return back to the

Fide Debug facility, and open a source rule file representing the Fide Inference Unit (FIU) of interest to you.

Display Data Flow

The Display Data Flow command initiates a dynamic display of data running through the nodes of your system. The data is superimposed on top of the schematic diagram. The numeric data values are displayed adjacent to the nodes to which they apply. They appear sequentially simulating control flow, tripping through the system from left to right. An example of the Data Flow Display is shown in Figure 6-32.

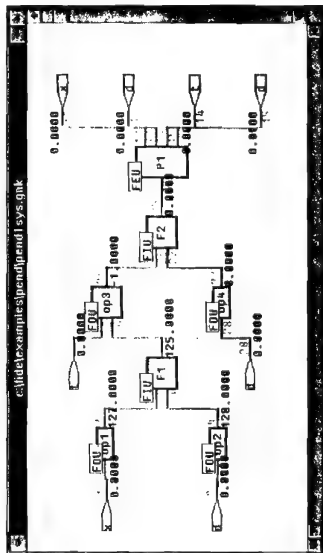


Figure 6-32 Composer/Display Data Flow Window

There are two options available with the Display Data Flow command: (1) Stepwise display, and (2) Continuous display. These options are provided as a hierarchical submenus to the main command. Choose Stepwise to control the display of data flow on a column by column basis. (Note: stepwise is *not* the same Step given in the Run Menu.) The Data Flow display shows data values of a single column of units or nodes at a time, after which you explicitly instruct it to go on to the next column or exit. Choose Continuous to

allow Composer to automatically cycle through continuous iterations of the data flow. A full cycle (or loop) through each column is considered a single step. To halt the data flow, press the ESC key.

Display Simulation

The Display Simulation command initiates display of a graphic plot representing the continuous values of input and output over a duration of time. Choose the Display Simulation command. A dialog box comes up prompting you for variables you wish to display. This dialog box is shown in Figure 6-33.

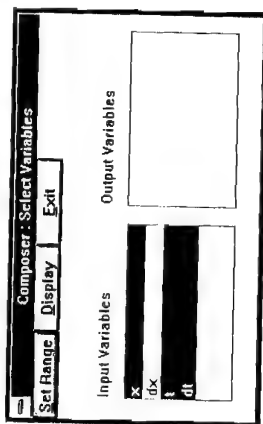


Figure 6-33 *Composer/Simulation/Select Variables Dialog Box*

You can select up to 7 variables. Click on the variable name, or use the up and down arrow keys to position the highlight bar, and then press the return key.

Set Range

For each variable displayed in either input or output list box, you must set a range of values by specifying the boundary points. Press the Set Range button located at the top left corner of the dialog box to initiate a set range dialog. The resulting dialog box is shown in Figure 6-34.

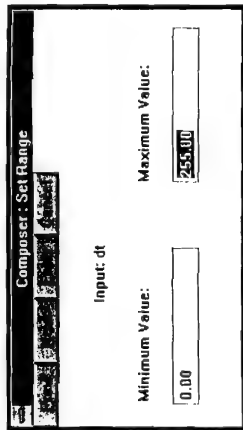


Figure 6-34 Composer/Simulation/Set Range Dialog Box

Display

Once you have made your selections and initialized the range, press the Display button at the top of the dialog box. The Composer Simulation display comes up. An example of the Simulation window is shown in Figure 6-35.

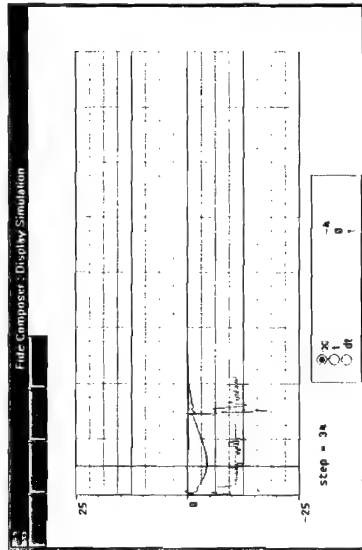


Figure 6-35 Composer/Display Simulation Window

Note that the values displayed are those of external system input and output variables. There is no provision for display of internal intermediate values between the units of a Fide inference system. This is intended, since an outside observer sees only the input and output values available to it.

Move the mouse into the graph region. As you move the mouse, the step position is dynamically displayed in text at the bottom of the graph. The variable values as a function of steps (time) are shown in the text box to the right of the graph. They too change dynamically as you move your mouse. Clicking on a variables radio button selects the variable and changes the vertical scale on the graph left boundary to reflect the dimensions of the selected variable.

<< Button

Press the "<<" button to load into memory an additional 512 steps of Simulation display. The Composer Simulator display can only display up to 512 steps. If simulations are longer than 512 steps, Composer buffers the non-visible portions of the display on disk.

>> Button

Press the ">>" button to load into memory a prior 512 steps of Simulation display. The Composer Simulator can only display up to 512 steps at one time. If simulations are longer than 512 steps, Composer buffers the non-visible portions of the display on disk.

Trace

The Trace command allows you to automatically launch the Fide Tracer function for any Fide Inference Unit you select in your system.

With a graphics format file open, choose the Tracer command. The tracer tool button (see Tools Buttons below) becomes highlighted. Move your mouse cursor to the Fide Inference Unit you wish to trace. Only FIUs are traceable.

Double click on the FIU symbol. You are brought into the Fide Tracer function with the Select Variable window of Tracer displayed. This window and the subsequent series of dialog boxes are described in chapter 4 under the Trace section of this User's Manual.

Note that if you are in the Stepwise Display DataFlow mode, the current values of your input and output variables are carried forward into the Trace function when you choose a particular FIU. This feature allows you to examine source statements in the context of a time varying simulation of your system.

Window

The Window menu is standard on all Multiple Document Interface (MDI) windows in Microsoft Windows 3.x applications. This menu provides functions to manage the multiple windows that can be brought up in an MDI environment. The Window menu is shown in Figure 6-36.

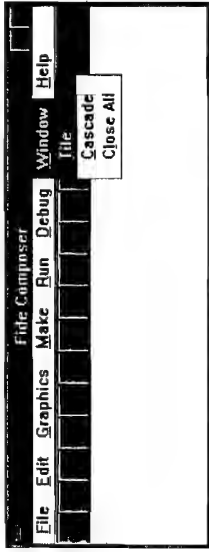


Figure 6-36 Composer/Window Menu

Tile

The Tile command causes all open windows in the main parent window to be resized and positioned so that all windows are visible and non-overlapping.

These windows are sized to fully fill the parent window. An example of tiled windows is shown in Figure 6-37.

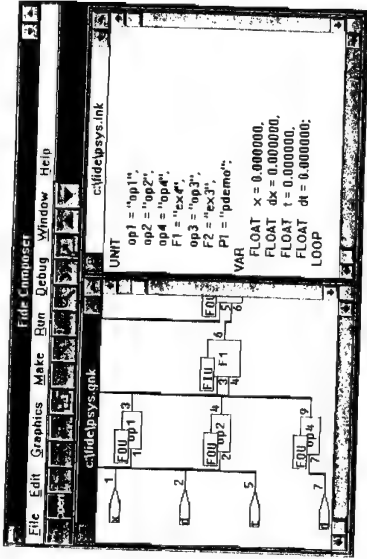


Figure 6-37 Composer/Tiled Windows

Cascade

The Cascade command causes all open windows in the main parent window to be positioned and sized in an overlapping sequence of windows. These windows are offset from one another in position so that the caption text of each window is visible. An example of cascaded windows is shown in Figure 6-38.

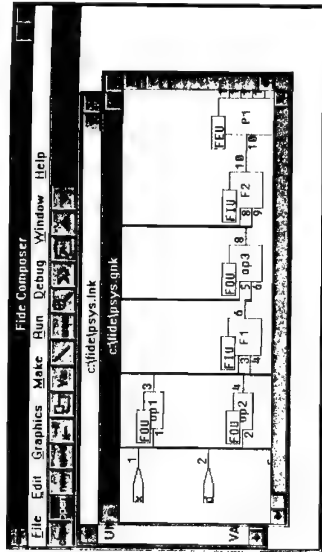


Figure 6-38 Composer/Cascaded Windows

Close All

The Close All command closes all windows, including those in minimized form, of a Multiple Document Interface (MDI) window. This command is a short hand way of selecting each window and explicitly closing each one by double clicking on the control menu or so other method.

Windows List

This region of the Window menu contains names of the windows open in the Multiple Document Interface environment. The currently active window (usually the topmost window) is indicated by a check mark at the left of the window name. Selecting any window name on the menu makes that window the active one. An example of a Window List is shown in Figure 6-39.

Help

The Help command brings up a Help window. The window contains a list of topics arranged alphabetically. Clicking on any one of these items will bring up material on the topic in its own separate window.

Tool Buttons

Composer provides a ribbon of buttons on the top row of the Composer window. These buttons perform identical functions to some of the most used commands in the Composer menus. The buttons allow you to access functions quickly and directly. The tool buttons and their equivalent menu command functions are given below. See the equivalent menu command explained in other sections of this User's Guide for a description of the function.

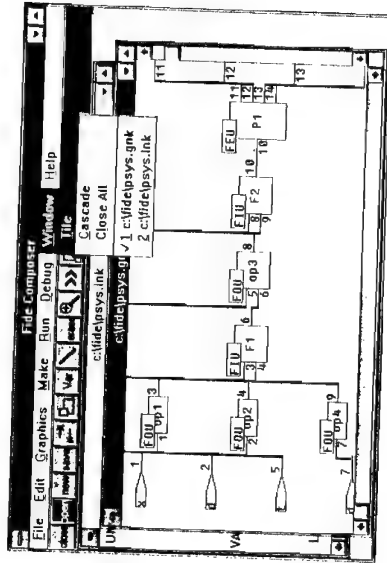


Figure 6-39 Composer/Window List



File/Close

File/Open

File/New

File/Save

Graphics/Draw Column

Graphics/Draw Unit

Graphics/Draw Variable

Graphics/Draw Line

Graphics/Erase

Graphics/Spy

Debug/Display Data Flow

Debug/Trace

Graphics/Zoom Out

Graphics/Zoom In

Fide generates a number of working files in the course of processing and analyzing data in fuzzy inference development sessions. In general, working files assume the filename of the source file on which a tool or process is being applied. The files are differentiated by filename extensions. This appendix lists the Fide file extension strings so that users can be aware of file naming conflicts that may arise. If the reader needs more information about Fide working files, contact Aptironix technical support.

Compiler

The Fide compiler generates and uses a number of working or intermediate files which are necessary to drive the multiple support functions available in Fide. A working file is named as follows:

SourceFileName.Ext

where *Ext* can be one of the following strings:

input: FILL
FDC

output: BRD
COD
EXI
EXO
ICD
MAP
OCD
RSP
SMF
TRC

ERR (only if errors occur)

Real Time Code Generator

The Real Time Code Generator uses and generates the following working files:

SourceFileName.Ext

where *Ext* can be one of the following strings:

- input: FIC
SME
SRS
- output: ASM

Composer

The Composer uses and generates the following working files:

ComposerSourceName.Ext
FideUnitSourceName.Ext

where *Ext* can be one of the following strings:

- input: MAP
GIR
LNK
- output: C
DAT
EXE
SIM

This appendix lists and explains the error and information messages that Fide can issue. They are listed alphabetically by functional category.

Some messages are listed as system messages. In these cases, close down Fide, restart Fide, recompile your source, and attempt the function again. If the problem persists, contact Apronix technical support.

Compiler

Fatal Errors

end of file found

End of file reached before termination of source statements encountered.

Possible errors: see message "end expected".

no rule

The compiler found no rule in a FIU program.

Possible errors: (1) is this intended as a FIU program? (2) any misused endSymbol? (3) any missing ifSymbol?

parser passed end of program

The compiler expected additional source input but encountered the end of program terminator.

Errors and Warnings

':' expected

A colon character expected in a source statement.

'=' expected

An equal symbol is expected here as a part of a calcClause.

'}' needed

Braces are used for rule-macros. Please check their pairing. In addition, a comma is missing or mistyped, the compiler may consider a brace missing and report this message.

'and' or '&' expected

An 'and' reserve word or '&' character expected in a rule statement.

'end' expected

All rules but the last one are terminated with a '::'. The compiler considers the last symbol of a rule statement read as being the rule terminator. If it is not, the compiler expects an endSymbol. An error message is generated if this is the not the case.

Possible errors: (1) endSymbol missing? (2) endSymbol immediately follows a digit? (3) is the intended symbol suppose to be a ':' or "&"?

'if' or ':' expected

An 'if' reserve word or ':' character expected in a rule statement

'is' or '=' expected

An 'is' reserve word or '=' character expected in a rule statement.

'then' or ';' expected

A 'then' reserve word or ';' character expected in a rule statement.

bracket needed

In an FIU program, the specification of the membership functions (if not singletons) must be enclosed in a pair of brackets. A pair of brackets should be found for each variable except: (1) the variable does not have an attached item such as engineering unit, range specification, and defuzzifier, or (2) the inference method is TVFI, and the variable is an output variable.

Possible errors: (1) misused symbol, (2) pairing of brackets, (3) is this an output variable and the inference method is TVFI? (4) is this an FIU program?

can not find input file

The data file (extension ".FDL") does not exist.

can not open input file

The data file (extension ".FDL") can not be opened.

Compile Success !!!

Fide source compilation was successful.

Currently activated file is not a data file

Data file compiler compiles only *.FDL files.

Currently activated file is not a source file

Source compiler compiles only *.FIL files.

file name missing

Compiler tried to read a DOS file name, but was unsuccessful.

Possible errors: (1) there must be a DOS file name after the feuSymbol; (2) the DOS file name should be enclosed in a pair of double quotation marks.
file not found

A required file is not available in the specified directory.

illegal or multiple definition

A variable is incorrectly defined.

in argument list

The compiler is trying to parse an argument list which is attached to a sameSpec, copySpec, operSpec, cutSpec, shiftSpec, or concSpec.

Possible error: (1) check format; (2) see message "not a label name".

238 Part II: Fide User's Guide

in defuzzifier specification

The compiler is trying to parse a singleton, but failed.

Possible errors: (1) an '=' or ':' is missing or mistyped; (2) this is intended to be a singleton or not?

in label list

An error exists in the variable label list.

incorrect header format

The compiler found the header not correctly formatted.

Possible errors: (1) spelling of reserved words; (2) incorrectly typed symbols.

incorrect number of I/O variables

There must be at least one input variable and one output variable. If you are using a tableClause, there must be exactly one output variable.

incorrect number of table items

The number of items in a table is different from what is determined by the table header based on parenthesis nesting.

Possible errors: (1) table header; (2) parenthesis nesting.

Input Compile Success !!!

Compilation of the Simulator Input File was successful.

label list missing

The label list of a variable can not be found.

missing clause head

The clause head can not be found in the source.

not a label name

A defined label name is expected here, but was not found.

Possible errors: (1) wrong symbol or label missing; (2) if a label name is intended here, and it is correctly spelled, check the definition of the label; (3) a symbol is misused before the current symbol.

not a legal label specification

The first symbol in a label specification is illegal.

not an input label name

A defined input label name is expected here, but was not found.

Possible errors: (1) wrong symbol or label missing; (2) if the label name is intended here, and it is correctly spelled, check the definition of the label; (3) a symbol is misused before the current symbol.

not an input variable name

A defined input variable name is expected here, but was not found.

Possible errors: (1) wrong symbol or label is missing; (2) if the variable is intended here, and it is correctly spelled, check the definition of the label; (3) a symbol is misused before the current symbol.

not an output label name

A defined output label name is expected here, but was not found.

Possible errors: (1) wrong symbol or label missing; (2) if the label name is intended here, and it is correctly spelled, check the definition of the label; (3) a symbol is misused before the current symbol.

not an output variable name

A defined output variable name is expected here, but was not found.

Possible errors: (1) wrong symbol or label is missing; (2) if the variable is intended here, and it is correctly spelled, check the definition of the label; (3) a symbol is misused before the current symbol.

out of memory

Compiling process requires more memory than is available.

parenthesis expected

The symbol last read is expected to be a parenthesis.

Possible errors: (1) is the parenthesis symbol missing or misplaced? (2) is there any left parenthesis misused before this position? check the pairing of parentheses carefully.

Please open a file before entering Compiler

Fide compiler requires an open and active source file before proceeding.

semicolon missing

Compiler expects a semicolon, but found none.

semicolon or comma missing

Compiler expects a semicolon or comma, but did not encounter it.

too many numbers

This error occurs if the compiler is parsing a listSpec.

Possible errors: (1) a symbol, especially decimal point, is mistyped; (2) a number prefixed by an atSymbol is out of the declared range; (3) the error is an implicit semantic error - refer to FID reference manual.

unknown operator

The operator in the headerClause of a FIU program is wrong.

unknown unit name

The unit name has not been previously defined.

unrecognizable character

An invalid character has been encountered.

unterminated string

A termination character was expected, but not encountered.

variable name misplaced

A variable name in a table is different from that determined by the table header based on parenthesis nesting.

Possible errors: (1) variable name spelling; (2) incorrect table header; (3) incorrect parenthesis nesting.

wrong expression

Error found in arithmetic expression.

Possible errors: (1) typing error; (2) is there any timesSymbol missing (e.g. you should type 2*a instead of 2a); (3) is there any misused reserved word in the expression?

wrong number

A number read as the last symbol by the compiler is incorrect.

Possible errors: (1) mistyping of the number; (2) if number is in the header, it may be too large or too small; (3) if number is the label specification, check range specification or grade specification; (4) an '@' is missing.

wrong parameter list

The encountered parameter list is invalid.

wrong range specification

The range specification in the varClause is not correct.

Possible errors: (1) check the format; (2) is the step length reasonable (compared with the interval)?

wrong statement

This message occurs either when the compiler failed to find an expected callClause/calClause or the format of the callClause/calClause is incorrect.

Possible errors: (1) spelling; (2) the feuSymbol/fouSymbol in the program header is incorrectly used.

MF-Editor**..MIBF file read error**

Attempted to read Membership Function file without success.

MF-Editor only works with FIU

Open or activate an FIU file before entering MF-Edit.

Not editable. Proceed anyway?

A membership function specification can only be edited in memory. It cannot be saved permanently to disk.

Please open a file before entering MF-Edit

The MF-Edit function requires an open and active source file before proceeding.

Debugger**A Debugger is still running**

A new instance of the Debugger program has been invoked when a current one is still running. Close down the currently running Debugger before initiating a new one. If you can not close down the current Debugger, close down Fide and restart Fide.

A debugging tool is in use

An instance of either the Tracer, Analyzer, or Simulator is still running when one of those functions is invoked anew. Close off the currently running Debugger function before initiating a new one. If you can not close down the current function, close down Debugger itself and restart Debugger.

Analyzer cannot connect with Debugger

The Analyzer program is attempting to communicate with the Debugger program through Dynamic Data Exchange (DDE), without success. Close down both the Analyzer and Debugger programs and restart each.

.COD file read error

Attempted to read Fide compiler object file without success.

Debugger can't work with selected engine

The selected MPU is not currently supported by Debugger.

Debugger cannot connect with debugging tool

The Debugger program is attempting to communicate with either the Tracer, Analyzer, or Simulator programs through Dynamic Data Exchange (DDE), without success. Close down both the Debugger and Debugger functions and restart each.

Debugger cannot connect with Fide

The Debugger program is attempting to communicate with the Fide program through Dynamic Data Exchange (DDE), without success. Close down both the Debugger and Fide programs and restart each.

Disk full !!!

Not enough disk space left to perform a Fide function requiring disk I/O.

.EXI file not found

Data file not compiled.

.EXO file not found

Data file not compiled.

Fide Debugger only works with FIU

Open or activate an FIU file before launching Debugger.

File not compiled

A Debugger function requires a compiled source unit before proceeding.

Initialization error

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

.MAP file read error

Attempted to read MAP file without success.

Memory allocation error

Attempted to allocate memory unsuccessfully. Probably not enough memory.

No data file. Please execute this unit.

An intermediate data file has not yet been created. The Execute command creates this file.

No .BRD file. Please execute this unit

The BRD file must exist before Analyzer can proceed. The Execute command creates this file.

No .EXO file. Please execute this unit

The EXO file must exist before Analyzer can proceed. The Execute command creates this file.

No FONT file in fide directory

A font file for vertical printing could not be found.

No input file !

The Simulator Input File could not be found.

No Label

A label must be selected before processing can proceed.

No .RSP file. Please execute this unit

The RSP (response) file must exist before Analyzer can proceed. The Execute command creates this file.

Number of selected variables exceeds 7

Only 7 variables may be selected for Simulation display in both the Simulator and Composer programs.

Please open a file before entering debugger

Debugger requires an open and active source file before proceeding.

Please plug the Hard Key to the parallel port

Fide protection feature requires the supplied hardware key to be connected to parallel port 1 before Fide can proceed.

Simulator cannot connect with Debugger

The Simulator program is attempting to communicate with the Debugger program through Dynamic Data Exchange (DDE), without success. Close down both the Simulator and Debugger programs and restart each.

System font is not found

A specified font could not be loaded into Windows memory.

Tracer cannot connect with Debugger

The Tracer program is attempting to communicate with the Debugger program through Dynamic Data Exchange (DDE), without success. Close down both the Tracer and Debugger programs and restart each.

246 Part II: Fide User's Guide

Run Time Code

Can not write .ASM file

RTC can not open FIU's MCU assembly file for writing (extension name is .ASM). This error occurs if the disk is write protected or the media is bad.

Can not write .BIN file

RTC can not open FIU's FP3000 format real time code file for writing (extension name is .BIN). This error occurs if the disk is write protected or the media is bad.

Can not open .COD file

FIU's code object file (file extension name is .COD) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed.

Can not open .CVT file

RTC can not find system data file (extension name is .CVT) for a given target chip code conversion. This error occurs if the path \fide\bin has not been set in your autoexec.bat file. Reinstall FIDE as a possible corrective solution.

Can not open FIDEFILE

RTC can not find system file named 'fidefile' in the \fide\bin directory. This error occurs if the path of \fide\bin has not been set in autoexec.bat file.

Can not open .FIU file

FIU's standard fuzzy inference unit data file (file extension name is .FIU) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

Can not open .MAP file

FIU's code object file (file extension name is .COD) can not be found in current directory. A possible reason is the FIU has not been compiled or

the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

Can not open .MBF file

FIU's membership function object file (file extension is .MBF) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

Can not open .SMF file

FIU's standard membership function data file (file extension name is .SMF) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

Can not open .SRS file

FIU's standard rule set data file (file extension name is .SRS) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

FIU initializing error

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

FIU read error

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

Illegal code size

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

Illegal defuzzifier

The defuzzification method can not meet the specification of a given target RTC generator. Check the FIU source code, change the defuzzifier according to RTC specifications, and recompile the FIU.

Run Time Code

Can not write .ASM file

RTC can not open FIU's MCU assembly file for writing (extension name is .ASM). This error occurs if the disk is write protected or the media is bad.

Can not write .BIN file

RTC can not open FIU's FP3000 format real time code file for writing (extension name is .BIN). This error occurs if the disk is write protected or the media is bad.

Can not open .COD file

FIU's code object file (file extension name is .COD) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed.

Can not open .CVT file

RTC can not find system data file (extension name is .CVT) for a given target chip conversion. This error occurs if the path \vide\bin has not been set in your autoexec.bat file. Reinstall FIDE as a possible corrective solution.

Can not open FIDEFILE

RTC can not find system file named 'fidefile' in the \vide\bin directory. This error occurs if the path of \vide\bin has not been set in autoexec.bat file.

Can not open .FIU file

FIU's standard fuzzy inference unit data file (file extension name is .FIU) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

Can not open .MAP file

FIU's code object file (file extension name is .COD) can not be found in current directory. A possible reason is the FIU has not been compiled or

the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

Can not open .MBF file

FIU's membership function object file (file extension is .MBF) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

Can not open .SMF file

FIU's standard membership function data file (file extension name is .SMF) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

Can not open .SRS file

FIU's standard rule set data file (file extension name is .SRS) can not be found in current directory. A possible reason is the FIU has not been compiled or the compilation failed. Also a non-FIU file could have been opened inadvertently. RTC only works with FIU files.

FIU initializing error

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

FIU read error

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

Illegal code size

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

Illegal defuzzifier

The defuzzification method can not meet the specification of a given target RTC generator. Check the FIU source code, change the defuzzifier according to RTC specifications, and recompile the FIU.

Illegal FIU type

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

Illegal fuzzy operator

Some fuzzy operator can not meet the specification of a given target RTC generator. Check the FIU source code, change the fuzzy operator according to RTC specifications, and recompile the FIU.

Illegal grade value

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

Illegal inference method

The inference method of FIU can not meet the specification of a given target RTC generator. Check the FIU source code, change the inference method according to RTC specifications, and recompile the FIU.

Illegal label number in SRS

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

Illegal membership function mode

The membership function mode can not meet the specification of a given target RTC generator. Check the FIU source code, change the membership function mode according to RTC specifications, and recompile the FIU.

Illegal membership function shape

The shape of membership functions is not allowed in a given target RTC generator. Check the FIU source code, modify the membership function according to RTC specifications, and recompile the FIU.

Illegal rule

This is a system error. If the error occurs frequently, contact Apronix Technical Support.

Illegal variable step length

The step length of some variables can not meet the specification of a given target RTC generator. Check the FIU source code, modify unreasonable variable step lengths, and recompile the FIU.

Illegal variable range

The range interval of some variables can not meet the specification of a given target RTC generator. Check the FIU source code, modify unreasonable variable ranges, and recompile the FIU.

No hard key

Hard key has not been plugged on the parallel port of PC.

No label found

No label is declared in some variables so that it can not meet the specification of a given target RTC generator. Check the FIU source code, declare at least 1 label for each variable, and recompile the FIU.

Not enough memory

Can not allocate memory successfully. Release other Windows applications or expand memory complement on your PC.

Please open a file before entering RTC

The Real Time Code (RTC) Generator requires an open and active source file before proceeding.

RTC Convert Success !!!

Real Time Code Conversion was successful.

Too large grade value

The grade value is too large so that it can not meet the specification of a given target RTC generator. Check the FIU source code, change the grade value according to RTC specifications, and recompile the FIU.

Too many input variables

Too many input variables have been declared in FIU, so that it can not meet the specification of a given target RTC generator. Check the FIU source code, decrease the number of input variables according RTC specifications, and recompile the FIU.

Too many labels

Too many labels have been declared in some variables, so that it can not meet the specification of a given target RTC generator. Check the FIU source code, decrease the number of labels according RTC specifications, and recompile the FIU.

Too many output variables

Too many output variables have been declared in FIU, so that it can not meet the specification of a given target RTC generator. Check the FIU source code, decrease the number of output variables according RTC specifications, and recompile the FIU.

Too many rules

Too many rules have been used in FIU, so that it can not meet the specification of a given target RTC generator. Check the FIU source code, decrease the number of rules according RTC specifications, and recompile the FIU.

Too many terms in membership function

Too many polygon edges have been defined for a membership function, so that it can not meet the specification of a given target RTC generator. Check the FIU source code, decrease the number of polygon edges according RTC specifications, and recompile the FIU.

Composer**An open composer is still running**

This message comes up if you try to start composer through Fide while there is an instance of composer still running in the system. If an error

occurs in Composer, and you try to start Composer, this message comes up. Close down Fide, and relaunch Composer through a new instance of Fide.

Can only trace an FIU

The Fide tracer can only trace an FIU. The Composer calls the Fide Tracer, so that it too is restricted to tracing only FIU's.

Composer cannot connect with Fide

If launch Composer other than through Fide, this message comes up.

Corresponding '.Ink' file is not a correct one

You tried to convert an incorrect text form program to a graphics form program.

Corresponding data file does not exist

Tried to display dataflow or simulation for a system, but have not compiled nor run the system with the dataflow or simulation option respectively.

Corresponding executable file does not exist

Try to run a not yet compiled system in the Composer.

Entered an invalid unit name

When creating a new unit, entered name for the new unit must be an identifier.

Entered an invalid variable name

When creating a new variable, entered name for the new variable must be an identifier.

Entered name is already used in the system

When creating a new variable or a new unit, entered name for the new item is already used in the system.

Expect ''

A FCL program must be terminated by the character '':

Expect '{'

In a calling statement (beginning with a unit name) of FCL, the unit name must be followed by the character '{'.

Expect '}'

Closing character corresponding to the '{' in a calling statement of FCL.

Expect ':'

FCL uses the character ':' to separate the left part and the right part of a sending statement. The statement sends the value of the variable in the left part to the variable in the right part.

Expect ';'

Character ';' was expected here, but not found. In FCL character ';' is the delimiter of statements.

Expect identifier

An identifier was expected here, but not found. In FCL each statement begins with an identifier.

Expect '='

Symbol '=' was expected after declaring a variable. FCL uses this symbol to assign an initial value to a variable.

Expect digital number

In the initial value part of declaring a variable, the value character string is not correct. FCL only accepts the value character string consisting of 0-9 and character '.'.

Expect key word 'DO'

'DO' keyword was expected here, but not found. FCL uses the 'DO' keyword to express a open loop.

Expect key word 'END'

The statement list of a FCL program is terminated by the 'END' keyword. Following the 'END' keyword is the program terminating character ';

Expect key word 'LOOP'

'LOOP' keyword was expected here, but not found. FCL uses the 'LOOP' keyword to express a close loop.

Expect key word 'UNIT'

'UNIT' keyword was expected here, but not found. FCL uses the 'UNIT' keyword to start the unit declaration section.

Expect key word 'VAR'

In the variable declaration part, the Composer conversion routines found no 'VAR' keyword.

Expect type specifier

A type qualifier for the variable was expected here, but not found. FCL only supports a floating point data type.

Link from unit sssss to unit ssssss will create a loop. Please use a variable between the two units

When drawing a line from one unit to another, or converting a text form FCL program to the graphic form program, Composer found a direct loop from unit to unit. FCL only allow you to form a loop through a system variable, and it only supports one level of looping. Embedded or nested loops are not allowed.

MAP file read error

The Composer read an incorrect map file.

No dataflow window is open

Composer tried to display a dataflow for a selected system, but the system was not opened in graphics form.

Number Of Selected Variables Exceeds 7

When displaying simulation, you can only select 7 variables (input variable or output variable) for a given display session.

Only 'gnk' or 'lnk' file is editable in Fide Composer

Functions of composer only work on a 'gnk' or 'lnk' file.

Redeclaration found

The FCL conversion routines found a redeclared identifier here. The identifier may be a unit name or a variable name.

Selected MAP file is not a Fide unit's MAP file

When creating a new unit, the selected file with the extension name ".map" is not one created by the Fide compiler.

Selected Unit has too many inputs. It can not be expressed in graphic form

When creating a new unit in graphic form (while using the graphic editor of Composer), the selected unit can only have 40 inputs at most.

Selected Unit has too many outputs. It can not be expressed in graphic form

When creating a new unit in graphic form (while using the graphic editor of Composer), the selected unit can only have 40 outputs at most.

Send to itself

A sending statement was found sending a value to itself (the left part variable is the same as the right part variable).

Some units not compiled

Converting from text to graphics, all units used in the text program must have already been compiled by the Fide Compiler. Otherwise, the Composer converter can not find the corresponding map files for the not yet compiled units.

Statement syntax error

The FCL conversion routines parse a statement and encounter a serious error. This is most commonly caused by a missing colon, a mismatched or missing bracket, or a missing semicolon on the previous statement.

There is a loop in the DO body

A dataflow loop was found in an open loop FCL program.

There is a never used unit in the program

At least one unit was found which was declared but never used in the statement list.

Undefined identifier

The FCL conversion routines encountered an identifier in the statement, which was not previously declared.

Unit ssssss has too many inputs. It can not convert to a graphic form

This message is issued when you convert a text form FCL program to a graphic form program, and there is some unit in the program having more than 40 inputs.

Unit ssssss has too many outputs. It can not convert to a graphic form

This message is issued when you convert a text form FCL program to a graphic form program, and there is some unit used in the program having more than 40 outputs.

Unit ssssss not compiled. Please use Fide compiler to compile it

Before compiling an FCL program or opening its graphic form, each unit used in the program must be compiled by the Fide source compiler.

Used identifier is not an argument

In a calling statement, an identifier was encountered on the left side of the character ':'. If the right side contains two identifiers, the second one must be a name of one of the arguments of the corresponding unit.

Used identifier is not a unit name

FCL conversion routines found that a calling statement did not begin with a unit name, or that, in the right part of a statement, of two identifiers, the first one was not a unit name.

Variable type not selected

You did not select an input or output type while creating a new variable in graphic form.

- .asm extension, 61
- .ink extension, 79
- About Fide, 113
- abscissa, 31
- accelerator keys, 109
- aggregation, 21, 22
- All Views, 56
- Analyzer, 53, 169
 - elevation bar, 179
 - Rotate, 177, 179
 - Select Precision, 172
 - visualizations, 169
- Analyzer tool, 52
- anomaly, 57
- ANSI C statements, 215
- antecedents, 51
- Apronix
 - Run Time Library, 68, 97
- Apronix Standard Data Structure, 97
- Apronix, contacting
 - BBS, 98
 - FAX, 98
 - voice telephone, 99
- Arrange Icon command
 - Fide, 121
- assembly code, 61, 62
- assertion, 7
- autoexec.bat path, 102
- automatic conversion of format, 202
- average, 32
- blank column, 208
- block diagram, 40, 52
- Borland C, 79, 82, 199
- Borland C and Turbo C compilers, 68
- bound difference
 - MF-Edit, 131
 - Tracer, 157
- bound sum
 - MF-Edit, 131
 - Tracer, 157
- C language applications, 194
- C language code, 96
- C Source Code, 81, 215
- Cartesian coordinate graph, 58
- Cascade command
 - Composer, 229
 - Fide, 120
- Center of Gravity, 26, 191, 192
- centroid, 31, 32
- change directories, 111
- channel, 8
- characteristic function, 3
- Clear All command
 - Composer, 205
 - Fide, 115, 122
- Close All command
 - Composer, 230
- closed loop temperature control system, 67
- code, 65
- Color bands, 179
- comment, 39
- compare.map, 72
- comparing unit, 34
- Compile, 44, 58
- Composer
 - buttons ribbon, 231
 - components, 198

- Continuous display, 224
- Create New Line window, 76
- Create New Variable, 72
- File menu, 195
 - graphics format, 195
 - New Text command, 196
 - Select Range window, 88
 - Stepwise display, 224
 - System object, 194
 - system requirements, 193
 - text edit functions, 196
- Composer Close command, 202
- Composer command, 194
- Composer graphics editor, 68
- Composer main menu, 68
- Composer schematic diagram, 209
- Composer Set Value window, 81
- Composer Simulation display, 226
- Composer tool bar, 66, 76
- Composer: Select Variables window, 88
 - condition part, 7
 - consequence part, 7
 - continuous Display DataFlow, 84
 - Contour view, 54, 56
 - controller chips, off the shelf, 189
 - cool air fan, 37
- Copy command
 - Composer, 205
 - Fide, 115
- Course, 55
- Cross Section command, 180
- Cross Section view, 56
- cur, 67, 74, 83
- current temperature, 67
- currently active window, 230
- Cut command
 - Composer, 204
 - Fide, 115
 - data flow path, 74, 75, 77
 - Debug, 58, 65
 - Debug command, 147
 - Debug menu
 - Composer, 223
 - Debug option, 45
 - Debug/Display Data Flow, 232
 - Debug/Trace, 232
 - Debugger main window, 53, 58
 - debugging tool, 42
 - Debugging using the Fide Analyzer, 52
 - Debugging using the Fide Tracer, 45
 - default, 74
 - default range, 86
 - default value, 81
 - defuzzification, 11, 15, 21, 22, 26, 31
 - des, 67, 72, 74, 75, 83
 - desired temperature, 37, 67, 81
 - desired temperature input variable, 72
 - Display button, 59
 - Display Data Flow command, 224
 - Display Data Flow view, 82
 - Display Simulation, 85
 - Display Simulation command, 225
 - DOS environment, 79
 - DOS version, 94
 - Draw, 48
 - Draw Column, 208
 - Draw Line tool, 211
 - Draw Unit, 209
 - Draw Variable tool, 210
 - Edit menu
 - Fide, 114
 - editing functions, keyboard/mouse, 118
 - editor
 - text, 109
 - embedded controller, 198
 - energy saver mode, 59, 67
 - Enter button, 59, 81, 88
 - Erase tool, 213
 - errors, 44
 - examples, 37
 - Execute
 - Analyzer, 171
 - Execute button, 59
 - execution unit, 34
 - Exit command
 - Fide, 113
 - external object modules, 199
 - fan_sys.gnk, 79
 - fan_sys.gnk diagram, 68
 - fan_sys.lnk, 79
 - fan_sys2.exe, 80
 - fan_sys2.gnk, 66
 - FANRULE1, 37
 - FANRULE2, 37, 52, 58, 65, 67, 71
 - FANRULE2.asm, 62
 - FANRULE2.FDL, 58
 - FANRULE2.map, 69
 - FANS, 58
 - fans, 67, 71, 77
 - FANS sub directory, 37
 - fast mode, 59
 - FCL, 96
 - FEU, 65, 67, 72
 - FEU module, 71
 - Fide
 - acronym, 93
 - exiting, 105
 - functions, 96
 - Help, 123
 - Installing, 99
 - linking function, 193
 - Quick Start Guide, 95
 - Reference Manual, 96
 - starting, 103
 - subdirectories, 100
 - tutorial, 95
 - User's Guide, 95
 - Fide Composer, 65
 - Fide Composer and Fide Library, 65
 - Fide Execution Unit, 193, 209
 - Fide Inference Language, 209
 - Fide Inference Unit, 193, 209
 - Fide Inference unit, 40, 65, 89
 - Fide inference units, 37
 - Fide inference units (FTU), 39
 - Fide library, 67
 - Fide Manuals
 - reference conventions, 105
 - typefaces, 97
 - Fide Operation Unit, 193, 209
 - Fide Reference Manual, 39, 40, 65, 68, 69
 - Fide Tracer, 89
 - Fide User's Guide, 48, 58
 - FIDEMAKE.CFG, 217
 - FIL, 96
 - file "filter", 111
 - file "mask", 111
 - Composer, 197
 - File ..New Graphics, 68

- Tracer, 157
- minimization, 19, 167
- minus, 67, 72, 75
- minus module, 83
- mod, 67
- mod system variable, 83
- modify, 57
- Modify button, 47
- module, 67, 77
- Motorola
 - chips, 148
 - code, 189
 - MCUs, 61
 - specifications, 191
- mouse requirement, 94
- Multiple Document Interface
 - Analyzer, 174
 - Composer, 228
 - Debugger, 148
 - Fide, 111, 119
 - NegativeBigDiff, 41
 - New command
 - Fide, 110
 - New Graphics command, 195
 - Next command
 - Composer, 207
 - Fide, 118
 - non-fuzzy inference units, 65
 - normal mode, 59, 67
 - object code, 141
 - Open As Graphics command, 197
 - Open As Text command, 200
 - Open command
 - Fide, 58, 110
 - Open coomand
 - Fide, 37
 - Operation, 31, 48
- optimum performance, 42
- output label
 - Mamdani, 138, 163
 - Tracer, 163
 - TVFL, 138, 163
- output membership function, 46
- Output Node, 200
- output variable, 1, 40, 59
- output variable clause, 42
- outvar, 40, 42
- Paste command
 - Composer, 205
 - Fide, 115
- pencil, 74
- point and click
 - MF-Edit, 132
- Polygon, 48
- polygon tool
 - MF-Edit, 134
 - Tracer, 158
- probability product
 - MF-Edit, 131
 - Tracer, 157
- probability sum
 - MF-Edit, 131
 - Tracer, 157
- processor, 94
- range, 41
- Real Time Code, 189
- Real Time MCU code, 61
- real world application, 65
- Real-time Code Generator, 61
- red arrows, 57
- Replace ... command
 - Composer, 206
 - Fide, 117
- reserved word, 40
- restore text, Fide text editor, 114
- RTC option, 61
- rule
 - evaluation, 11, 13, 22
 - rules, 43
- Run, 82, 85
- Run menu, 219
- run mode, 67, 81
- Run option, 80
- Run_mode, 54, 59
- Save As command
 - Fide, 112
- Save As Graphics command, 202
- Save As Text command, 203
- Save as Text option, 78
- Save command
 - Fide, 112
 - scales, color matched, 186
- Search ... command
 - Composer, 205
 - Fide, 116
- Select Input
 - Composer, 221
- Select Precision dialog box, 55
- Select Variable
 - Analyzer, 169
 - MF-Edit, 127
 - Simulator, 182
 - Tracer, 150
- Select Variable window, 88
- Set Margin
 - Tracer, 158
- Set Range, 86
 - Composer, 225
- Set Steps command, 222
- Set Value
 - Analyzer, 170
- Composer, 219
- Simulator, 184
- Tracer, 151
- Set Value option, 81
- side view, 54
- Simulating system behavior, 84
- simulation, 65
- Simulator, 58
 - Select Variables Window, 59
 - temporary data, 185
 - simulator, 34
 - simulator graph, 86
 - SIN extension, 221
 - singleton, 4, 29, 42
 - software simulation, 67, 68, 81, 82, 85
 - source code, 38, 67
 - source code listing, 41
 - Source File command, 142
 - source statements, 141
 - special unit, 34
 - Spy tool, 213
 - step, 89
 - step domain, 185
 - Stepwise option, 83
 - Stepwise or Continuous display
 - Data Flow, 83
 - support, 33
 - point, 4
 - value, 4
 - Surface view., 56
 - Switch to Composer, 65
 - syntax, 39
 - syntax of FTL, 43
 - system chart, 36
 - system diagram, 66, 84
 - system input variable, 83

- system input variable module, 74
- system variable, 67, 72
- Target menu, 148
- test, 65
- text diagram, 84
- text file, 79
- Text format
 - Composer, 197
- three dimensional display, 175
- three-dimensional surface, 54
- Tile command
 - Composer, 228
 - Fide, 119
- Trace
 - Analyzer, 178
 - Composer, 227
 - Simulator, 186
- Trace button, 58, 60
- Trace option, 89
- trace, scope of, 164
- Tracer, 48, 60, 150
 - Select Output Label window, 49
 - Select Variable window, 48
- Tracer command, 150
- Tracer tool, 45
- transfer function, 169
- true perspective, 181
- truth value, 3, 29, 42
- tune, 65
- tuning tool, 42
- Turbo C, 79, 82, 199
- TVFI, 21, 29, 31, 32, 42
- TVFI Inference Method, 139
- Type of Variable box, 72
- Undo command
 - Composer, 204
- Fide, 114
- Unit Name box, 69
- Untitled.gnk, 69
- Untitled.gnk system, 73
- Using the data flow viewer, 82
- Using the Fide Simulator, 58
- Using the graphics editor, 68
- Variable Name box, 72
- version number, 113
- weighted average, 15, 29, 31
- what if scenarios, 169
- Window List
 - Composer, 230
 - Fide, 122
- Window menu
 - Fide, 119
- Windows, 94
 - non-overlapping, 119
 - overlapping, 120
- Windows clipboard, 115
- Writing and Compiling source code, 37
- Zoom In command, 214
- Zoom Out command, 214